FAST WALSH—FOURIER TRANSFORM GENERATOR

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of

MASTER OF TECHNOLOGY

by

R. VENKATESAN

to the

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR
JULY, 1976

C E R T I F I C A T E

This is to certify that the project work entitled 'FAST WALSH-FOURIER TRANSFORM GENERATORS' has been carried out entirely under my supervision and has not been submitted elsewhere for a degree.

R Biswas

R.N. Biswas

Assistant Professor
Department of Electrical Engineering
Indian Institute of Technology
KANPUR

EE-1976-M-VEN-FAS

A C K N O W L E D G E M E N T

I take this opportunity to express my deep gratitude towards Dr. R.N. Biswas who not only suggested this challenging and interesting problem, but also was a source of constant inspiration and encouragement. I am deeply grateful to him for those moments when he brought me on a right track, while I was thoroughly lost in the vastness and complexity of the problem. This thesis would not have been completed without his consistent intellectual supervision.

I am also very thankful to my friend Mr. A.S. Kanade for the fruitful discussions I had with him at the beginning of my work.

I gratefully acknowledge the help rendered by many of my friends especially Mr. R. Vadivelu and Mr. K.N. Gopalan, during the various stages of this work.

Lastly but never the least, I would like to thank Mrs. B. Rukmini Devi for her excellent job of typing.

R. VENKATESAN

LIST OF CONTENTS

LIST OF SYMBOLS

r	DFT Frequency
N	Number of Samples
A_r	r^{th} coefficient of DFT
D_1	First data
D_2	Second data
Δr	Increment
a	real part of first data
b	imaginary part of first data
c	real part of second data
d	imaginary part of second data
e	real part of weight
f	imaginary part of weight
g	real part of increment
h	imaginary part of increment
M_1, M_2, M_3, M_4	Multipliers
ADD	Adder
SUB	Subtractor
EOC	End of computation
n	Number of iterations
I	Iteration number
W^r	Trignom tric weight

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The Fourier Transform and the Fourier series are important concepts in the field of signal analysis and spectral characterisation. Discrete Fourier Transform (DFT) is a transform of its own right such as Fourier integral transform or the Fourier series transform. It defines a spectrum of time series and so DFT of a time series of equally spaced samples is closely related to Fourier Transform and so DFT is mainly useful for power spectrum analysis and filter simulation on digital computers. If the time series consists of N samples, then about N^2 real multiplications and additions are required to compute the coefficients.

Fast Fourier Transform (FFT) is a highly efficient algorithm developed by Cooley and Tukey[1] for computing the DFT of a time series. It takes the advantage of the fact that the calculation of the coefficients of the DFT can be carried out iteratively which results in a considerable savings of computation time. For the same number of samples N, it requires only $N\log_2 N$ arithmetic operations as compared to N^2 in DFT.

Discrete Walsh Transform (DWT), like DFT, can also be used to define the spectrum of time series, and requires considerably less computation, since the Walsh functions which constitute a complete set of orthonormal functions like sine and cosine functions, can only take the values 1 and -1, thus eliminating multiplication totally. In Walsh transforms, different signals are characterised by their sequency spectrum rather than by the frequency spectrum. Especially in the case of sampled data systems the descriptions of sampled signals in the sequency domain is advantageous since the sequency spectrum of such signals is finite.

Fast Walsh Transform[2] (FWT) is also an algorithm making the computation of the Walsh Transform very fast. It has been the subject of several studies in the past few years even though the basic theory was established as early as 1958 by Good[3]. The FWT has an inherent computational advantage over the FFT. The FWT requires only real addition and subtraction operations while the FFT requires complex multiplications. This difference results in a significant hardware simplicity advantage and a possible speed advantage for the FWT.

The sequency spectrum is dependent on the phase relation between the signal and the sampling signal. With the audio signals, the phase of the signal is relatively unimportant for carrying information, but the phase completely changes the sequency spectrum. This should be compared with a frequency spectrum of audio signal, where the majority of information is obtained in the amplitude of the spectrum and very little in the phase of the components. This makes FFT a more useful tool than FWT for audio applications.

Video signals on the other hand have two characteristics that make them compatible with Walsh Transform. The transient characteristics (and hence the phase) of video signal is important and they can be represented exactly by a Walsh Transform. Also a video signal is divided into time slots by the line scanning process, and this is a natural subdivision for a Walsh Transformation. So the Walsh Transform is a specific example of generalised harmonic analysis approach to image classification whereas Fourier Transform is useful for audio signal processing. Moreover FWT can lead to a powerful, yet relatively simple Digital Walsh Filters which can be used for image processing where image enhancement and data compression can be achieved by suitable filters. Two dimensional Walsh Transform has already been used for picture transmission in Japan and it will not be far off from now to see 3-dimensional picture transmission using 3-dimensional Walsh Transform.

Kanade[4] developed an Arithmetic Unit for FFT using two's complement serial arithmetic, where the hardware basically consists of four multipliers, one adder and one subtractor along with associated registers and control circuits. He has also presented an 'Increment Logic' for the weight generation in FFT where the ROM required for storing the weights in the conventional method has been replaced by a smaller ROM for just storing the increments. This thesis is an extension of the above work, the objective being (i) to incorporate the generation of the trigonometric weights in the FFT AU itself using counter-decoder synthesis, (ii) to build a separate AU for FWT and (iii) to combine the control of both FFT and FWT in one control unit. The weight generation scheme and AU for FFT are described in the second chapter, followed by a description of the FWT AU, its hardware and operation in the third chapter. The fourth chapter deals with the principles and the hardware realisation of the main control unit to control the address and data flow in both FFT and FWT.

CHAPTER II

ARITHMETIC UNIT FOR FFT

FFT is a clever computational technique of sequentially combining weighted sums of data samples so as to produce the DFT coefficients. The mathematical background of FFT and the arithmetic unit, as well as the scheme for the generation of the trigonometric weights are discussed in this chapter.

2.1 Mathematical background of FFT

DFT of a time series of equally spaced samples is closely related to the Fourier Transform of the corresponding continuous signal. The r th coefficient of DFT is given by

$$A_r = \sum_{k=0}^{N-1} X_k . W^{kr}, \qquad r = 0, 1, \ldots . \; N-1, \tag{2.1}$$

where X_k is the k th sample of time series which consists of N samples, and

$$W^r = \exp(- \frac{2\pi jr}{N}) \quad \ldots \tag{2.2}$$

is the trigonometric weight.

As described previously, FFT is nothing but the Cooley - Tukey algorithm which is based on the property of factoring the record of samples, taking the DFT of the factors and then combining the results. For an N-point FFT, where $N=2^n$, n being an integer, there are n iterations, each iteration consisting of $\frac{N}{2}$ basic computations, resulting in a total of $\frac{N}{2}\log_2 N$ computation cycles. Each such cycle consists of a typical pattern called the 'butterfly pattern' which is nothing but the generation of a two-point transform according to the transform equations

$$D_1' = D_1 + W^r.D_2$$

$$\text{and} \qquad D_2' = D_1 - W^r.D_2$$

where D_1 and D_2 are complex numbers fetched from the sample locations in the memory and D_1' and D_2', the new values generated by the transformation, which are to be loaded back in the same sample locations. This basic cycle requires one complex multiplication, one complex addition and one complex subtraction. The actual hardware realisation for this arithmetic operation is schematically shown in Fig. 2.1 where $D_1 = a+jb$, $D_2 = c+jd$ and $W^r = e+jf$. For an N-point transform $\frac{N}{2}$ complex weights are required. The sequence of operation is illustrated by the signal flow graph for an 8-point FFT in Fig. 2.2. From the diagram it is obvious that each butterfly requires a pair of addresses and these addresses follow a repetitive logical pattern and the sequence of the addresses is different in each iteration.

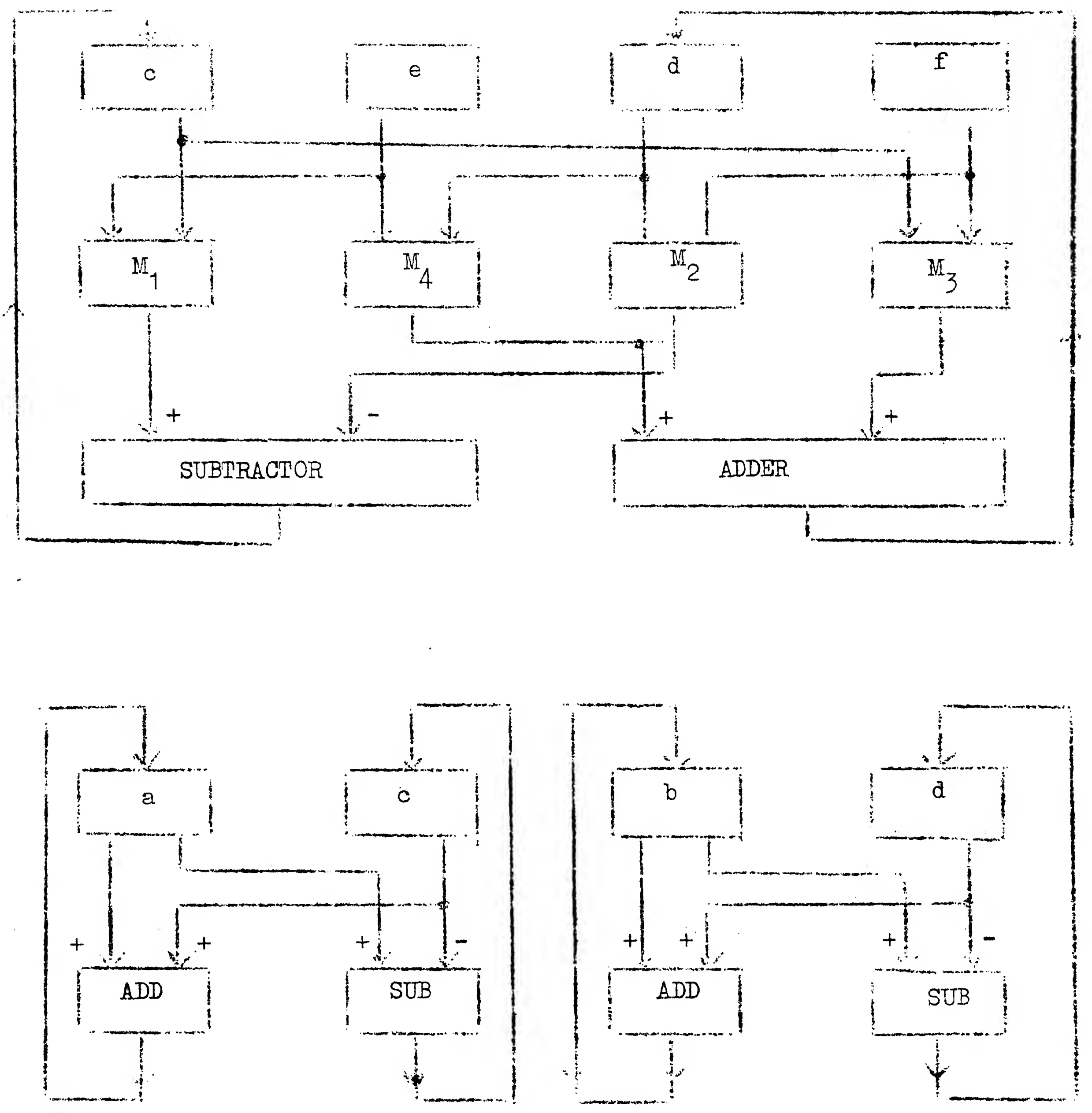


Fig. 2.1 FFT AU CONTROL FLOW DIAGRAM

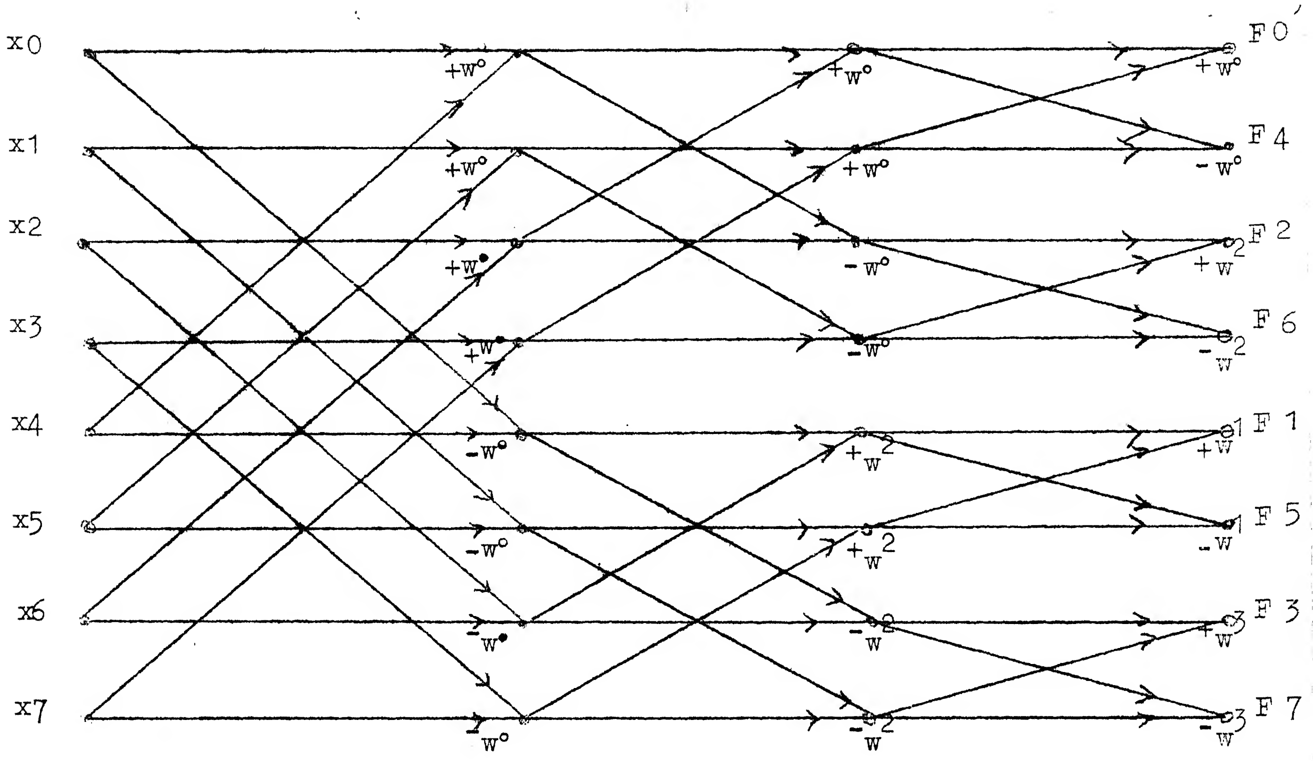


Fig. 2.2   SIGNAL FLOW GRAPH FOR 8-POINT FFT

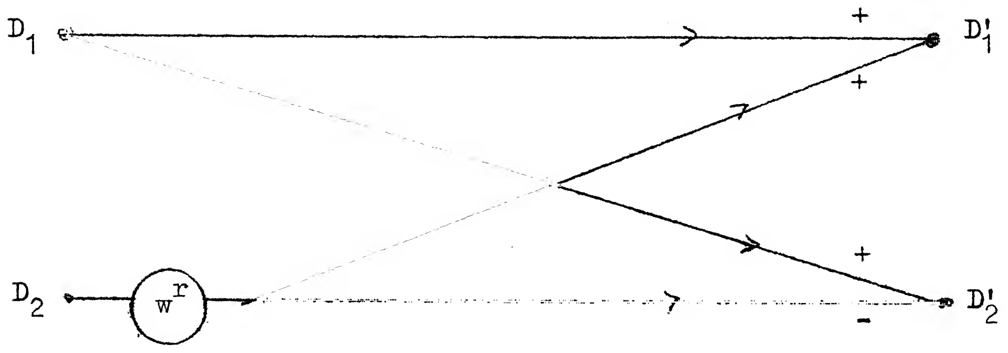


Fig. 2.3   BASIC BUTTERFLY COMPUTATION

2.2 'Increment Logic'

It can be generalised from the signal flow graph illustrated in Fig. 2.2 that all the weights are used only in the last iteration and the pattern in which they occur remains the same for all earlier iterations. Hence as far as the sequence of occurrence of weights is concerned, it is sufficient to analyse only the pattern in the last iteration. For an N-point transform, $\frac{N}{2}$ complex weights are required, and the pattern of the occurrence of these weights in the last iteration is given in Table 1, which also shows the increments Δr in the DFT frequency leading to the next required weight.

The increment $\Delta r = \frac{N}{4}$ means that the ratio of the new weight to the previous weight is given by

$$\exp\left(- \frac{j\pi}{2} \right) = -j$$

and hence the new weight can be generated by just interchanging the real and imaginary parts of the previous weight and changing the sign of the imaginary part. No new value of increment is thus necessary for this. Let us denote the successive distinct values of increment Δr, leaving out $\Delta r = \frac{N}{4}$ by k_i, where $i=1,2 \; . \; . \; n-2$. Then the general expression for k_i can be obtained from the inspection of Table 1 as

$$k_i = -\frac{N}{2} + \frac{3N}{2^{i+2}} \, , \; i = 1,2 \; . \; . \; . \; n-2 \qquad (2.3)$$

TABLE 1

Pattern of the Occurrence of Weights in an N-point FFT in the Last Iteration

Computation Number	r	Δr	Distinct value of Δr
1	0	-	
2	$\frac{N}{4}$	$\frac{N}{4}$	
3	$\frac{N}{8}$	$\frac{N}{8}$	
4	$\frac{3N}{8}$	$\frac{N}{4}$	$\frac{N}{4}, -\frac{N}{8}, -\frac{5N}{16}, -\frac{13N}{32}, \ldots$
5	$\frac{N}{16}$	$-\frac{5N}{16}$	
6	$\frac{5N}{16}$	$\frac{N}{4}$	
7	$\frac{3N}{16}$	$-\frac{N}{8}$	
8	$\frac{7N}{16}$	$\frac{N}{4}$	
9	$\frac{N}{32}$	$-\frac{13N}{32}$	
⋮	⋮	⋮	

TABLE 2

Pattern of Occurrence of Distinct Increments

Number of Points	Number of Iterations	Distinct Weights	Distinct Increments	Sequence of Distinct Increments
8	3	2	1	k_1
16	4	4	2	$k_1\ k_2\ k_1$
32	5	8	3	$k_1\ k_2\ k_1\ k_3\ k_1\ k_2\ k_1$
⋮	⋮	⋮	⋮	⋮
$N = 2^n$	n	$\frac{N}{4}$	$n-2$	$k_1\ k_2 \ldots k \quad .. \ k \quad k$

The pattern in which these distinct increments are required in the last iteration is shown in Table 2 for various values of N. The same pattern follows for any other iteration with appropriate truncation of the sequence.

Eqn. (2.3) shows that for the first increment in each iteration, $k_i\theta$ lies in the fourth quadrant, while for other increments it lies in the third quadrant. Hence the imaginary part of $\exp(-jk_i\theta)$ is always positive and its real part is always negative except for the first increment. Moreover, only n-2 distinct increments are required for a 2^n-point FFT. If the number of points is doubled, then only one more distinct value of the increment is required, as compared to $\frac{N}{4}$ distinct weights required in the conventional method. For each computation cycle, the new weight can be generated by the AU itself, which is capable of doing complex multiplication, complex addition, complex subtraction, only if the appropriate value of $\exp(-jk_i\theta)$ is available. The additional algorithm for FFT using this 'Increment Logic' is given in Fig. 2.4 in the form of flow chart, where

$$\exp(-jk_i\theta) = g + jh$$

This additional computation has to be performed before starting the usual butterfly computation shown in Fig. 2.3 for generating the required new weight. However, this additional step can be skipped for those additional computation cycles where the new weight is either the same as that in the preceeding cycle or can be generated from the latter by interchanging the real and imaginary parts.

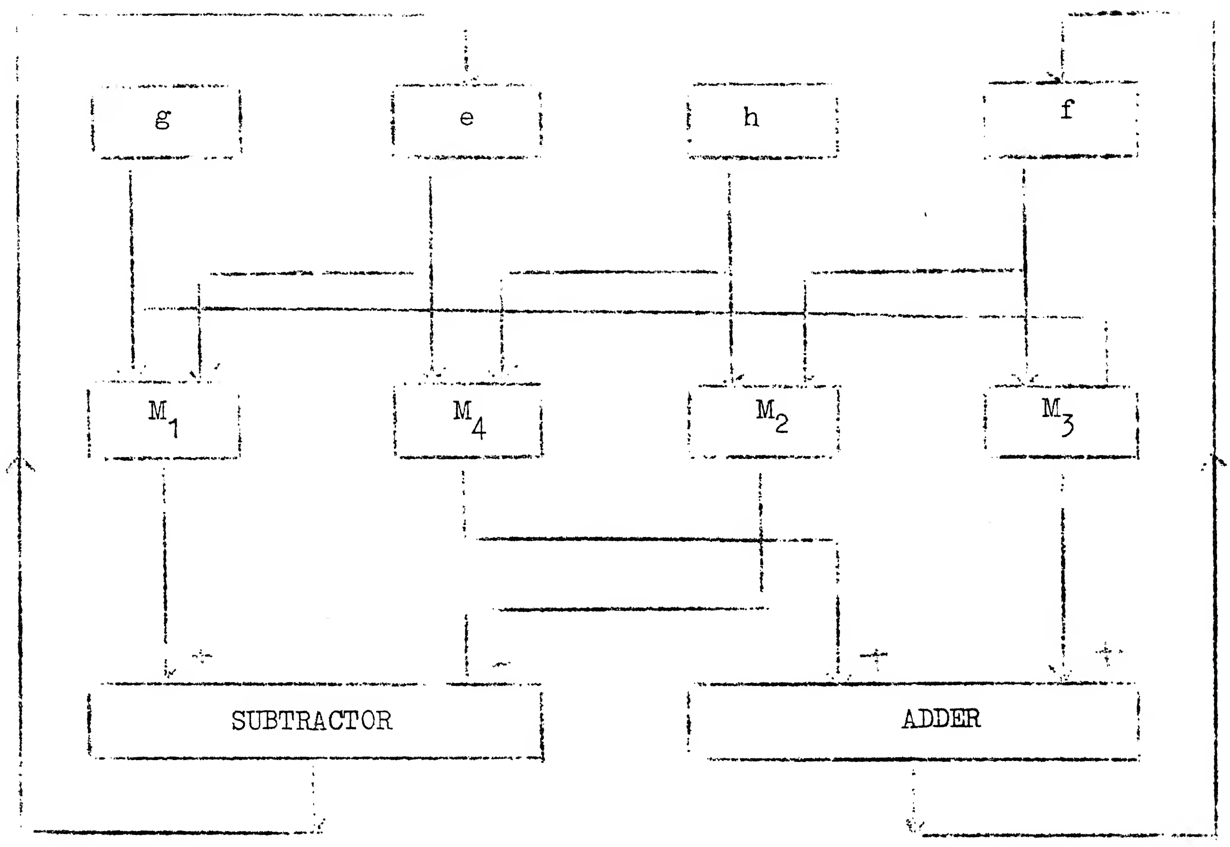


Fig. 2.4 ADDITIONAL AU CONTROL FLOW DIAGRAM FOR NEW WEIGHT GENERATION

2.3 Computation Counter

A binary computation counter is used to count the number of computations performed in each iteration, and this counter overflows at the end of each iteration. For a 2^n-point FFT, 2^{n-1} computations constitute each iteration and n-1 bits are therefore required for this counter. For a 2K-point transform, an 10-bit counter is required. This counter is just a ripple counter with the output of each flip-flop connected as the clock input of the next. The overflow of a particular bit is detected using an AOI gate corresponding to the required number of points of FFT/FWT.

2.4 Choice of Word Length

The number of bits required to represent the word should be chosen for the necessary resolution, and this number should be compatible with the number of bits required to represent the weight. The different values of distinct weights should differ at least by 1 bit in any discrete m-bit representation for the latter to be meaningful for a given value of N. From Table 1, it can be seen that $k_i\theta$ tends to $-\frac{N}{2}$ as i increases. As a result, the difference between the arguments of successive distinct factors $\exp(-jk_i\theta)$ used in the generation of new weights tends towards zero as i increases. It is therefore necessary to ensure that the bit representation is at least capable of distinguishing the last two increments. Moreover, if a minimal overall accuracy has to be guaranteed, the distinct increments should have even greater resolution.

Now, from eqn. 2.3,

$$k_{n-3} - k_{n-2} = 3 \tag{2.4}$$

Hence the ratio between the last two distinct factors used in the generation of weights is given by

$$F = \frac{\exp(-jk_{n-3}\theta)}{\exp(-jk_{n-2}\theta)} = \exp(j3\theta) \tag{2.5}$$

For large values of N_1, $\theta \rightarrow 0$ and hence

$$F = 1+j3\theta = 1 + \frac{j6\pi}{N} \quad \ldots \tag{2.6}$$

Let the required resolution demand that the smallest change in k_i should produce an effect at least in the last x bits in the computed value of a distinct new weight. Clearly, then, for an m-bit representation to be compatible with a 2K-point FFT, F must differ from unity at least in the last x bits of the imaginary part. This requires the following condition to be satisfied.

$$\frac{6\pi}{N} \geqslant 2^{(x-m+2)} \quad \ldots \tag{2.7}$$

as the magnitude has to be represented by m-2 bits, allowing 1-bit for sign and 1-bit for overflow, or

$$m \geqslant \log_2 N + x + 2 - \log_2(6\pi) \geqslant n + x - 2 \quad \ldots \tag{2.8}$$

Table 3 gives the minimum requirement for 1 percent resolution.

2.5 Interchange Command Generation for a 2K-point FFT

It has been seen in section 2.2 that in some cases the new weight can be generated from the previous weight by giving the 'Interchange Command'. From an extension of Table 1, the pattern in which this command occurs is easily obtained as shown in Table 4 in terms of iteration number and the computation number. The latter may be represented by the states Q_0^C , Q_1^C , Q_2^C . . . . (where Q_0^C represents the LSB, Q_1^C the next significant bit and so on) of the flip-flops constituting the computation counter. Each state of the computation counter corresponding to a particular combination of values of Q_0^C , Q_1^C . . . . is henceforth referred to as 'CC setting

2.6 Increment Command Generation for a 2K-point FFT

This command is generated whenever a new weight has to be generated from the previous weight using the 'Increment Logic'. There are n-2 distinct increments and the pattern in which they occur for a 2K-point FFT in terms of I and CC setting is given in Table 5.

'X' indicates don't care condition, with the qualification that no command should come when the CC setting is 0000000000.

The patterns shown in Table 4 and 5 can be easily extended to arrive at the following generalised rule about the occurrence of the interchange and increment commands in terms of the state transitions of the computation counter.

TABLE 3

Word Length for 1 percent Resolution

N	n	m
32	5	10
512	9	14
2048	11	16

TABLE 4

Pattern of Occurrence of Interchange Command

I	CC Setting										No. of Commands
	Q_9^C	Q_8^C	Q_7^C	Q_6^C	Q_5^C	Q_4^C	Q_3^C	Q_2^C	Q_1^C	Q_0^C	
1	No Command Necessary										0
2	1	0	0	0	0	0	0	0	0	0	1
3	X	1	0	0	0	0	0	0	0	0	2
4	X	X	1	0	0	0	0	0	0	0	4
5	X	X	X	1	0	0	0	0	0	0	8
6	X	X	X	X	1	0	0	0	0	0	16
7	X	X	X	X	X	1	0	0	0	0	32
8	X	X	X	X	X	X	1	0	0	0	64
9	X	X	X	X	X	X	X	1	0	0	128
10	X	X	X	X	X	X	X	X	1	0	256
11	X	X	X	X	X	X	X	X	X	1	512

'X' indicates 'don't care' condition

TABLE 5

Pattern of Occurrence of Increment Command

I	CC Setting										No. of Commands
	Q_9^C	Q_8^C	Q_7^C	Q_6^C	Q_5^C	Q_4^C	Q_3^C	Q_2^C	Q_1^C	Q_0^C	
1	NO COMMAND NECESSARY										0
2	NO COMMAND NECESSARY										0
3	X	0	0	0	0	0	0	0	0	0	1
4	X	X	0	0	0	0	0	0	0	0	3
5	X	X	X	0	0	0	0	0	0	0	7
6	X	X	X	X	0	0	0	0	0	0	15
7	X	X	X	X	X	0	0	0	0	0	31
8	X	X	X	X	X	X	0	0	0	0	63
9	X	X	X	X	X	X	X	0	0	0	127
10	X	X	X	X	X	X	X	X	0	0	255
11	X	X	X	X	X	X	X	X	X	0	511

INTERCHANGE AFTER EVERY 0 to 1 TRANSITION OF Q^C_{n-I} (I=2,3..n)

INCREMENT AFTER EVERY 0 to 1 TRANSITION OF Q^C_{n}-I+i

(I = 3,4, . . . n, i = 1,2 .. n-2)

where $i \leq I-2$

It is obvious that the interchange command can be looked upon as the special case of increment commands for i=0.

From the foregoing rules for the generation of different commands, it is clear that an interchange command is to be generated for every 0 to 1 transition of Q^C_{n-2} in the second iteration and so on. The exact logic for the generation of interchange command can be obtained in terms of the settings of the iteration register which is used to keep track of the iteration currently in progress and the computation counter as follows.

Let $A_1, A_2, \ldots A_n$ be the n-line output of the iteration register (i.e. $A_I = 1$ only in the Ithiteration and 0 for other iterations) and let C_{INT} be the interchange command variable given by the following Boolean expression

$$C_{INT} = A_2 \cdot Q^C_{n-2} + A_3 \cdot Q^C_{n-3} + \cdots A_I\, Q^C_{n-I} + \cdots A_n\, Q^C_0 \ldots \qquad (2.9)$$

It is then obvious that an interchange command is to be generated for every 0 to 1 transition of C_{INT}.

Similarly, the following expression can be obtained for the increment command variable C_{INC}, whose 0 to 1 transitions will call for the increments

$$C_{INC} = A_3 \; Q^C_{n-3} + A_4 \; Q^C_{n-4} + \ldots\ldots A_n \; Q^C_0 \; . \; . \; . \qquad (2.10)$$

2.7 Precision and Speed Considerations

The total number of increment commands in a 2^n-point FFT can be obtained using Table 5 as

$$M_{INC} = 2^{n-1} - n \; . \; . \; . \qquad (2.11)$$

and the total number of butterfly computations involved is given by

$$M_{BUT} = n.2^{n-1} \; . \; . \; . \qquad (2.12)$$

Each butterfly computation and each increment involves a complex multiplication and each multiplication has got an error which is due to the truncation in multiplication. This error can be considered as a 'noise' propagated throughout all the iterations. The magnitude of this error may increase or decrease when it is in propagation. The ratio of the total number of increment commands to the total number of computations is given by

$$\frac{M_{INC}}{M_{BUT}} = \frac{2^{n-1} - n}{n.2^{n-1}} \simeq \frac{1}{n} \qquad (2.13)$$

For large values of n. For a 2K-point FFT, this ratio is $\frac{1}{11}$ which is rather small, and so additional error involved in the system due to the increment logic can be considered to be small.

Though the new weights are generated by the Arithmetic Unit itself, the total time of this FFT for all iterations can be made the same as the conventional method of storing the weights in the ROM, where the weights are readily available when the computation is started. In the 'Increment Logic' also weights can be made readily available when the computation is started, provided the AU is sufficiently fast, utilising the time taken at the end of each computation to write the data back in the memory and to read the new data. Since the memory cycle time is not too small, we can trigger the AU to generate the new weights as soon as the computations are over. Before the new data are read, the AU can finish the generation of new weight and is ready for the next computation. No overlapping is possible between these as data and weights are in separate registers.

2.8 Block Diagram

The block diagram for FFT AU is shown in Fig. 2.5. In this block diagram, the adders, subractors, multipliers, the associated registers and the control circuits have already been made by Kanade[4]. The additional hardware required for the modified version of the weight generation, AU clock generation, data display etc. has been presented in the following sections.

2.9 Counter-Decoder Scheme for Interchange/Increments

Interchange command can be generated by having a simple decoder decoding for I and CC setting in accordance with the pattern shown in Table 4. The decoder for the increment command should have (n-2) output

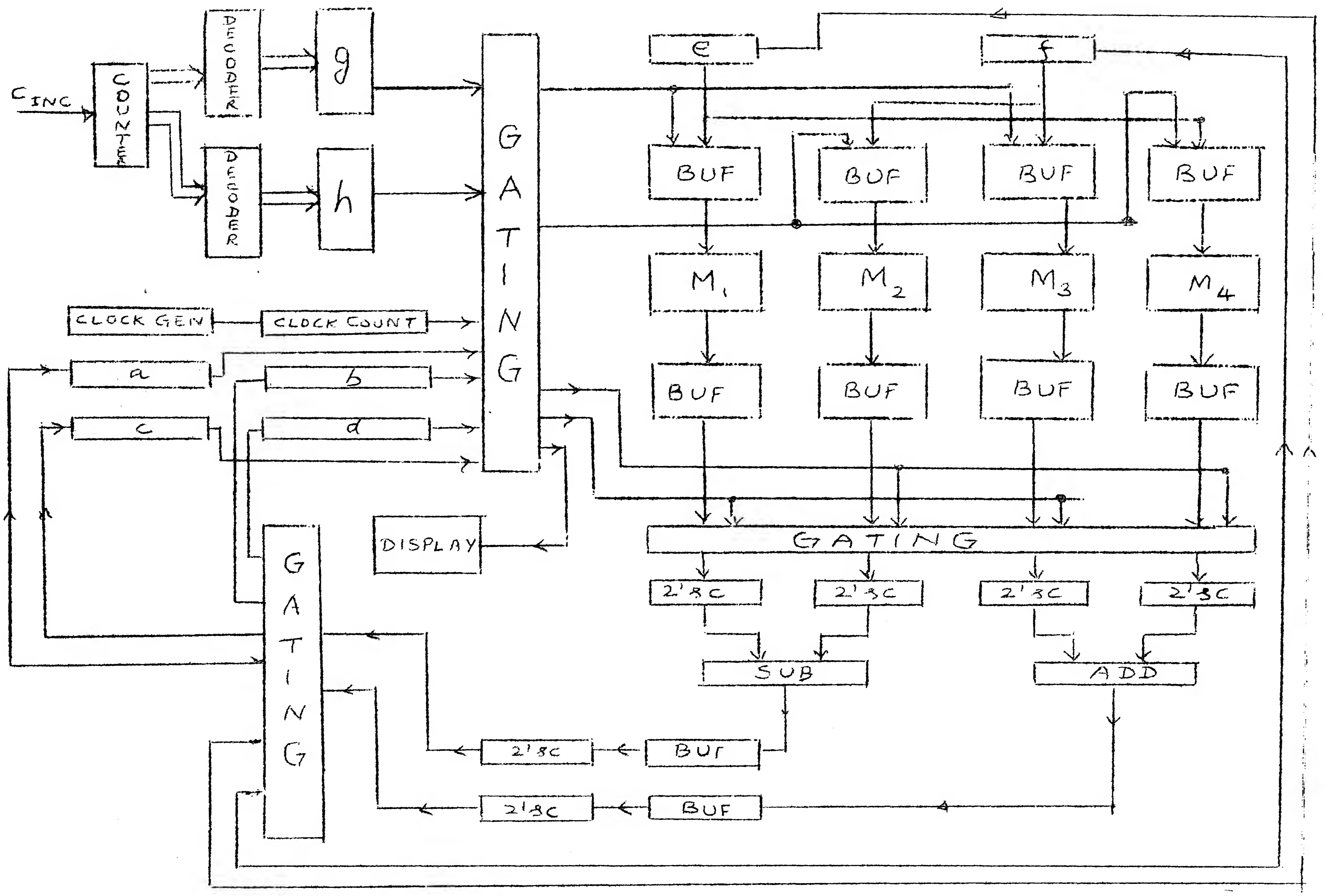


Fig. 2.5 BLOCK DIAGRAM FOR FFT

lines corresponding to (n-2) distinct increments. From these lines, it is possible to generate the numerical values of $\cos(-k_i\theta)$ and $-\sin(k_i\theta)$ using an 'Increment Generator'. Even though this is possible in principle, the size of the decoder becomes too large as it is a multioutput, multi-input decoder, with (n-2) line output and n lines from the iteration register and (n-1) lines from the computation counter as inputs.

Alternatively, we can use a sequential decoder in which an increment counter is used to count all the increment commands corresponding to various distinct increments and from the appropriate transition of this counter we can generate (n-2) different lines. This increment counter is just a ripple counter in which 0 to 1 transition of C_{INC} is used as the clock. It is obvious from Table 1 that the ith distinct increment is to be generated whenever Q^{I}_{i-1} makes a 0 to 1 transition. This scheme is shown schematically as a block diagram in Fig. 2.6.

In this scheme, the decoders for C_{INT} and C_{INC} are both single output decoders which are made as per the Tables 4 and 5 respectively and they are shown in Fig. A 2.1. Since only one of 'n' output variables of the iteration register is '1' in a particular iteration, the decoders can be realized in very simple AND-OR structures.

The increment counter is shown in Fig. A 2.2 where the top row of the flip-flops constitute a ripple counter which just counts the number of increment commands and the bottom row of the flip-flops register the 0 to 1 transition of the corresponding flip-flop in the top row. When

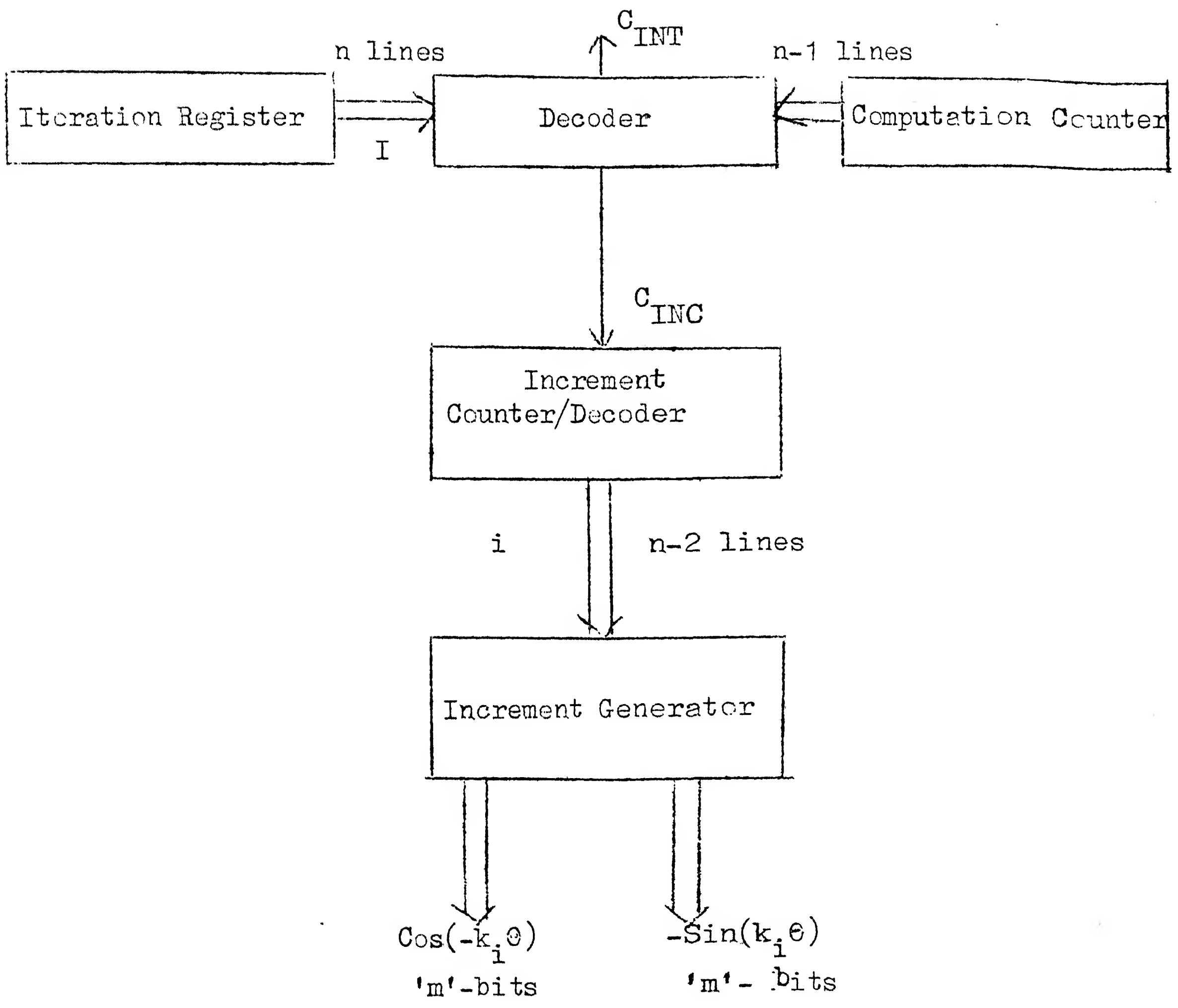


Fig. 2.6 BLOCK DIAGRAM FOR INTERCHANGE/INCREMENT

distinct increment k_i is to be generated then only the i^{th} flip-flop in the bottom row is '1' and all the remaining other flip-flops in that row are zero. All the flip-flops in the bottom row are cleared by the increment command itself which goes to zero at the end of the computation. The top rows of the flip-flops are clearèd at the end of each iteration.

2.10 Increment Values for a 2K-point FFT

The number of distinct increments to be generated is 9(=n-2) for a 2K-point FFT. The values of $\cos(-k_i\theta)$ and $-\sin(k_i\theta)$ corresponding to these 9 distinct increments k_1, k_2 . . . k_9 have been tabulated in Table 6, using a 16-bit sign-magnitude binary coding with the sign bit appearing as the most significant bit. It is evident from Table 6 that most of the bits in the real and imaginary parts are 1's and 0's respectively. A simple decoder therefore can be easily designed to generate the appropriate values of $\cos(-k_i\theta)$ and $-\sin(k_i\theta)$ which appear as two simultaneous outputs of the decoder having a 9-line input representing the value of i. The circuit diagrams for the above decoders for real and imaginary parts are shown in Figs. A 2.3 and A 2.4 respectively, where the shift registers are used for parallel to serial buffers since the serial arithmetic has been used in FFT AU.

Table 6

16-bit Representation for the Values of the Real and Imaginary Parts of $\exp(-jk_i\theta)$

i	$k_i\theta$	$\cos(k_i\theta)$	$-\sin(k_i\theta)$
1	$-\frac{\pi}{4}$	0 101101010000001	0 101101010000001
2	$-\frac{5\pi}{8}$	1 011000011111101	0 111011001000001
3	$-\frac{13\pi}{16}$	1 11010100110110	0 10001110001110
4	$-\frac{29\pi}{32}$	1 11110100111110	0 01001010010100
5	$-\frac{61\pi}{64}$	1 11111101001110	0 00100101100100
6	$-\frac{125\pi}{128}$	1 11111111010011	0 00010010110101
7	$-\frac{253\pi}{256}$	1 11111111110100	0 00001001011011
8	$-\frac{509\pi}{512}$	1 11111111111101	0 00000100101101
9	$-\frac{1021\pi}{1024}$	1 11111111111111	0 00000100010110

governed by a control algorithm which has 12 steps, each step consisting of 16 pulses, as the word length has been chosen as 16 bits for 2K-point FFT.

The control algorithm consists of 3 basic sub-cycles of 4 steps each. The first sub-cycle is used for generating the new weight from the previous weight using 'Increment Logic'. The complex multiplication of the second data with the weight is done in the second sub-cycle and the basic butterfly operation is performed in the third sub-cycle.

It may be sometimes necessary during the testing of the equipment to start the operation at any sub-cycle and perform the computations corresponding to one full sub-cycle or just one step in a sub-cycle. This leads to a necessity of generating a burst of 64 pulses or 16 pulses respectively when a manual trigger for A.U. is given. In the normal 'RUN' mode this trigger for the burst is governed by the main control unit itself.

For faster AU operation, it may have to start from the first step itself when increment command is present or from the fourth step if the interchange command is present or from the fifth step if none of the commands is present. To provide all these facilities, synchronous presettable counters are used where the presetting is done by the above commands in such a way that the burst starts from the desired step.

A gated oscillator using Schmitt Trigger NAND gates is made as the clock generator and the first clock pulse of this is used to preset the counter. For the actual computation, this clock pulse should be disabled and only the second and the onward pulses are used. This provision is made by a 7473 J-K flip-flop with suitable gates as shown in Fig. A 2.5.

To generate the burst of 16 or 64 or the whole set of pulses, a 7495 shift register is used where the presetting is done in the required way. The shift register overflows at the end of required number of clock pulses and this overflow resets a flip-flop which stops the clock generation. The burst is started by setting this flip-flop by a manual trigger or the AU trigger from the control unit. Provisions for external clock required to test the AU step by step are also made by using NAND gates which select either the internal or external clock. The timing diagram for the above circuit is shown in Fig. A 2.6.

2.12 Buffer for Multipliers

There are four serial-parallel multipliers in the FFT AU designed by Kanade using the 'add and shift' algorithm for multiplication. These multipliers use only the magnitude parts of the multiplier and the multiplicand for multiplication and so the 16th bit of the multiplier and multiplicand should be made zero before going to the multiplier. Since each multiplier bit controls 16 bits of the multiplicand, buffers should be added to increase the fan out capacity. In each multiplier output, the LSB comes in the output line and the remaining bits of the product come in a separate line. Some form of gating is therefore essential to combine all the bits in the proper order in a single line. D flip-flops with suitable gates and buffers are used for this purpose and the circuit realisation for the above one is shown in Fig. A 2.7.

2.13 Display

For testing the FFT AU, some form of display of data is essential when manual triggering is used to test each stage of the AU. Since each data consists of real and imaginary parts of 16 bits each, two sets of 16 LEDs are used for display. Since serial arithmetic is used, a serial to parallel buffer is also provided to make the data available in parallel for display . Since two sets of data, one trigonometric weight and one increment are involved, AOI gates are used to select one of the above four sets. The circuit diagram for the above one is shown in Fig. A 2.8.

CHAPTER III

FAST WALSH TRANSFORM GENERATOR

Walsh Transform is quite useful in the applications of image processing and pattern recognition. This transformation is binary in nature, making it suitable for implementation in a special purpose computer. A second advantage of FWT over FFT arises out of the fact that the FWT algorithm requires only $N\log_2 N$ additions, where N is the number of samples, and no multiplication. In this chapter, the mathematical background of FWT and hardware aspects are described in detail.

3.1 Mathematical background of Walsh Functions

Walsh functions are periodic with period N where N is an integral power of two. So the complete orthogonal set will have N distinct functions. These functions are designated as Wal (m,n). The complete set is represented over the range m = 0,1,2 ... N-1 and n = 0,1,2 ... N-1.

The first two discrete Walsh functions are defined as

$$\mathrm{Wal}\ (0,n) = 1 \qquad \text{for } n = 0, 1 \ldots\ldots N-1$$

$$\mathrm{Wal}\ (1,n) = 1 \qquad \text{for } n = 0, 1, 2 \ldots N-1$$

$$-1 \qquad \text{for } n = \frac{N}{2}, \frac{N}{2} + 1, \ldots N-1$$

The remainder of the set can be generated by an iterative equation

$$\mathrm{Wal}\ (m,n) = \mathrm{Wal}\ \left[(\tfrac{m}{2}), 2n\right] \cdot \mathrm{Wal}\ \left[m-2(\tfrac{m}{2}), n\right] \ldots \qquad (3.1)$$

where $(\frac{m}{2})$ indicates the integer part of $\frac{m}{2}$.

3.2 Fast Walsh Transform

Given an N-point* real arrayf(n), we can define the Walsh Transform as

$$F(m) = \sum_{n=0}^{M-1} f(n)\ \mathrm{Wal}\ (m,n) \qquad m = 0, 1, \ldots\ldots N-1 \qquad (3.2)$$

and inverse transform as

$$f(n) = \frac{1}{N} \sum_{m=0}^{N-1} F(m)\ \mathrm{Wal}\ (n,m) \ldots\ n = 0, 1, \ldots N-1 \qquad (3.3)$$

Since Walsh Transforms can have values of +1 and -1 only, computations of Eqn. (3.2) require no multiplications. Using Eqn. (3.1), a computational algorithm can be easily derived analogous to the Cooley - Tukey, algorithm. This algorithm will require $N\log_2 N$ summations to compute a complete Walsh Transform rather than N^2 as indicated by Eqn. 3.3. The derivation of this algorithm parallels the one given by Cooley - Tukey and hence this algorithm is called Fast Walsh Transform (FWT).

* In FWT literature the term 'N-length' is generally used instead of 'N-point'. However, to have a uniform notation, the term 'N-point' is used in this thesis for FFT as well as

3.3 Signal Flow Graph

The signal flow graph for an 8-point FWT is given in Fig. 3.1a. The flow graph is similar to the one given in Fig. 2.2 differing only in the weights. From this signal flow graph, one can convert some Cooley - Tukey transform programs to Walsh transform by setting all the trigonometric values to 1.0 or by removing the steps which multiply the array values by the trigonometric values. Also, the complex part of the Cooley - Tukey operation can be removed since the weights are all real. The inverse Walsh transform is identical to the Walsh Transform except for all values are divided by N.

The final result A_3 will contain all the coefficients of 8-point Walsh Transform. However, just as in Cooley - Tukey algorithm the order of values will be in bit reversed form.

3.4 Block Diagram for FWT

The organisation of the Fast Walsh Transform Generator is essentially the same as that of a special purpose computer. It basically consists of 4 units namely the I/O unit, the memory, the arithmetic unit and the control unit. The analog signal to be transformed is digitised using an A/D converter at the input and these samples are stored in a Random Access memory (RAM). This RAM is used for storing the data, intermediate results and the final results. The task of AU is to carry out the computations, and the main control unit controls the processor and the memory. Main control unit also generates the addresses for the basic computations. The block diagram for FWT is shown in Fig. 3.2.

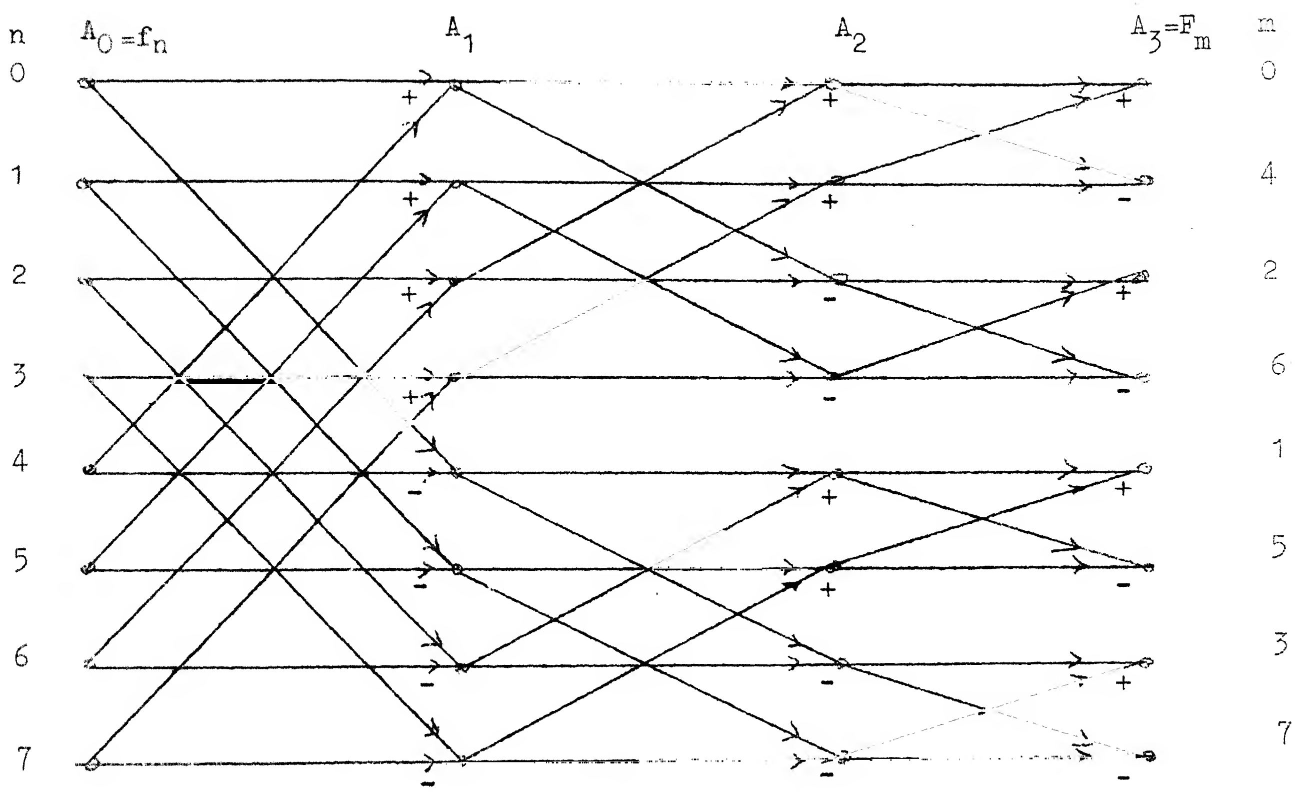


Fig. 3.1a SIGNAL FLOW GRAPH FOR 8-POINT FWT

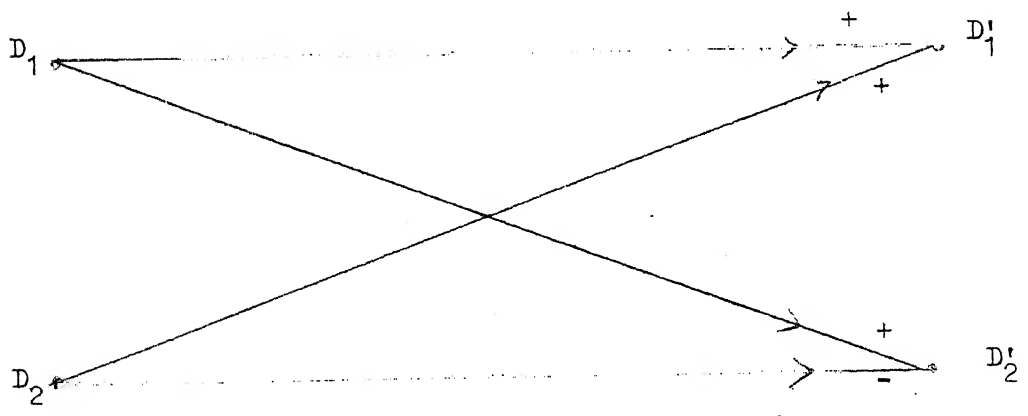


Fig. 3.1b BASIC COMPUTATION

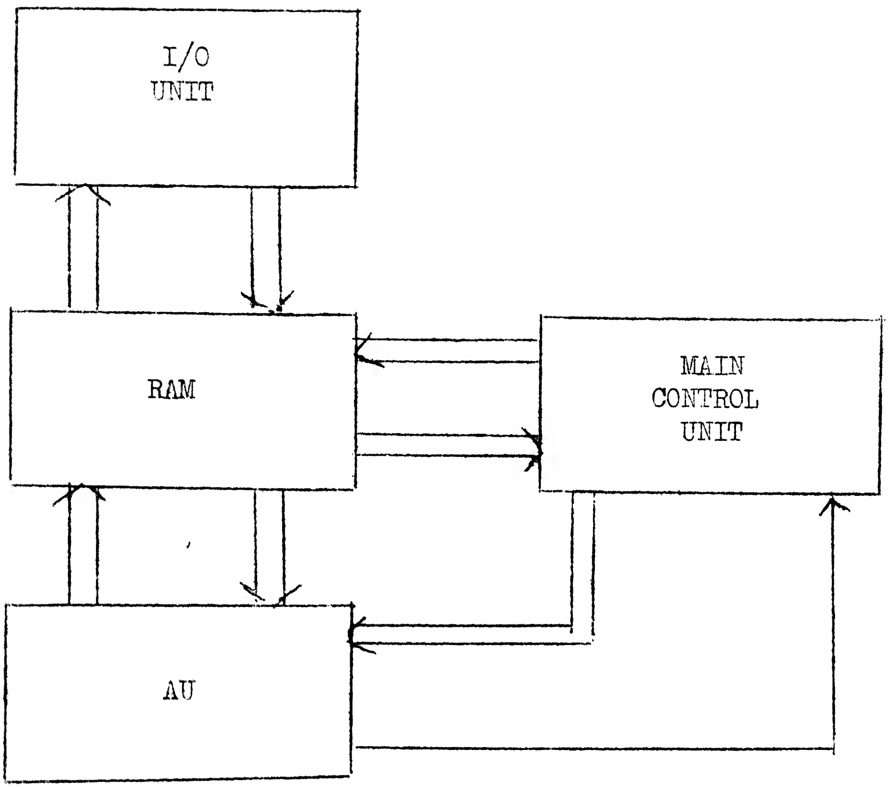


Fig. 3.2 BLOCK DIAGRAM FOR FWT

3.5 The Basic Computation

The FWT algorithm is similar to the Cooley - Tukey algorithm where for the number of samples $N = 2^n$, the factoring is continued till there are $\frac{N}{2}$ groups of 2 points. These two-point transforms can be calculated easily and they are combined to produce four-point transforms; four-point transforms are then combined to produce eight-point transforms and so on. If this process is continued, then after $\log_2 N$ such steps, the complete 'N' point transform is obtained. These steps are called iterations and in each iteration there are $\frac{N}{2}$ basic computations.

The basic computation of FWT is now the generation of two-point transform. It consists of three steps: (i) accessing two numbers from the memory locations, (ii) adding the two numbers and storing the sum in the first location and (iii) subtracting the second number from the first number and storing the difference in the second location. This basic computational pattern is shown in Fig. 3.1b where D_1 and D_2 are the data fetched from the first and second locations respectively, and D_1' and D_2' are the new values generated by the transformation, which are to be loaded back in the same locations.

3.6 1's Complement Versus 2's Complement

Basically the FWT AU consists of an adder and a subtractor with proper AU control. These operations can be performed in serial or parallel form. Parallel operations are naturally faster than the serial operations.

Since the arithmetic involved in FWT is not much, parallel processor has been chosen even though it is somewhat costlier than the serial processor. Since subtractions are also involved in the system, either 1's complement or 2's complement arithmetic has to bee~~n~~ chosen. 1's complement is evidently the better choice processor in this case, since the hardware required is much more for 2's complement in a parallel processor. Since the processor is fast, we need not have separate adders and subtractors as one of them can do the other function also by just changing the sign bit thereby minimising the hardware.

3.7 FWT Arithmetic Unit

The hardware realisation for FWT basically consists of a 16-bit parallel 1's complement adder which is shown in Fig. A 2.9. The two banks of exclusive - OR (EX-OR) gates at the data inputs of the adder are mainly provided for obtaining the 1's complement of the two data which are in sign-magnitude form when they are read into data registers from the memory. A bank of EX-OR gates at the output is used to obtain the 1's complement of the result and to bring the resultant data back in the sign-magnitude form before they are written back in the memory. 4-bit binary full adders 7483 are used for adders.

Since 1's - complement operation is used, final carry output of the adder is given as the carry input of the full adder corresponding to the LSB. This FWT AU is very fast as it is asynchronous and the total time it takes for one particular operation depends upon only the gate delays. The addition and subtraction are performed in accordance with the sign bit of FWT which goes to one of the inputs of the EX-OR gate corresponding to the MSB of the second data. Since the memory access time corresponding to one data is longer than the actual AU time for one operation, the memory cycle itself can be made to include the AU cycle also. During this cycle, the sign bit for the second data is switched in such a way that both addition and subtraction are performed for a computation. This memory cycle will be stopped only when all the iterations are over and so the total memory cycle time for all these iterations is the same as the total time that has been used to perform the Walsh Transformation.

CHAPTER IV

FWT AND FFT MAIN CONTROL UNIT

The main control unit generates the addresses for the butterfly computations involved in FFT and FWT and also it controls the data flow from FWT and FFT AU's to the memory. It also keeps track of the iteration number and computation number currently in progress. It also gives provisions in the FFT mode for scrambling the data to make decimation in time or decimation in frequency whichever is required. Modularity is also introduced in the control unit so that the same unit works for any number of points in FFT or FWT.

4.1 Iteration Register

The main function of the Iteration Register (IR) is to keep track of the iteration currently in progress. For an $N(=2^n)$ point FWT/FFT, the total number of iterations required is n and so a shift register of n bits can be used. Since the iteration number controls the other circuits like Address counter and decoders for increment/interchange command generation, a straight binary counter with n states is not preferable. The IR is designed in such a way that k th bit is '1' in the k th iteration and the rest of the bits are zero. Modularity can be achieved by loading a single '1' in a particular place of the shift register when the computation starts. Whenever an iteration ends, this single '1' is shifted one place to the right. When this '1' is shifted out of the shift register, it signifies all the n iterations to be over and the process stops.

4.2 Address Generation

Both in FFT and FWT, each computation requires a pair of addresses. These addresses follow a logical pattern as given by Kanade[4]. In the k th iteration, the k th bit of the first address is always '0', and the corresponding bit of the second address is always '1'. If this bit is suppressed, then the remaining bits are always in a natural binary sequence. So the second address can always be generated from the first address. So an Address counter of n bits for a 2^n-point FFT is required to generate this first address.

A simple ripple counter with n flip-flops can be used for this purpose. The clear terminals of these flip-flops are connected to the inverter of the corresponding IR bit. Thus if the k th bit of IR '1', then the k th flip-flop is cleared for the complete duration of the k th iteration, thus making the k th bit of this address permanently zero throughout that iteration. Instead of connecting the output of each flip-flop only the clock input of the next flip-flop as in a binary ripple counter, it is also connected to the clock input of the flip-flop after the next one in the line through a set of NAND gates as shown in Fig. A2.10. In k^{th} iteration, these gates cause the output of $(k-1)^{th}$ flip-flop to be connected to the clock input of $(k+1)^{th}$ flip-flop thus skipping the k^{th} flip-flop entirely. In other flip-flops, these gates cause output of the previous flip-flops connected to the clock input of the next thereby maintaining the natural binary sequence.

Since only k^{th} bit of the IR is '1' in k^{th} iteration, the individual bits of the second address can be obtained simply by means of a set of a 2-input OR gates, having the output of the address counter and the corresponding bit of IR as their inputs.

Since FFT contains both real and imaginary parts, the address can at most have 11 bits for 2K-points, while all the 12 bits can be used for the address in FWT, as there is no imaginary part present in the data.

4.3 Address Scrambling

For a naturally ordered sequence of samples, the Fourier coefficients and the Walsh coefficients do not occur in the natural order but in the bit reversed order. To access a particular coefficient, we must find the reversed bit integer of the address and access the coefficient from that location. For example, in the 8-point FFT, the coefficient F(3) occurs in location 6 because 6(110) is the reversed bit integer of 3(011). So the scrambling operation means the reallocating the data from its present address to a location whose address is given by the reverse-bit ordering of the present address.

For FFT, the above process is called 'decimation in frequency'[5, 6] since the Fourier coefficients occur in the reversed bit order. On the other hand, if the Fourier coefficients are to be obtained in the natural sequence of frequency, then the data or the record of samples must be scrambled in the reverse bit order before the start of the computation. This is called 'decimation in time'. So in general, if decimation in frequency is required then the scrambling operation should be done after all the computations are over and if 'decimation in time' is required, then the scrambling operation should be done before the actual start of the computations.

In the case of FWT, however, scrambling does not tend to an ordering of the coefficients according to sequency, but leads to what is known as 'dyadic ordering'[7]. To achieve sequency-ordering for FWT, the addresses have first to be treated as being in the Gray code, and converted therefrom to binary before scrambling is done. This not only requires extra computation time, but also means that sequency-ordered FWT cannot be obtained using FFT algorithm simply by bypassing the multiplications. Thus from the view point of the fast transform, dyadic ordering is advantageous.

Since a modular design for 512, 1024, 2048 and 4096 is aimed, AOI gates are used in an appropriate manner to do the bit reversal operation as shown in Fig. A 2.12. While scrambling, the operation should be continued only upto half the number of points after which the operation should be stopped as further scrambling will bring the coefficients once again in the bit reversed address as they were before scrambling.

Since the RAM permits access to only a single word at a time, one - of - two selectors are used to select one of the two addresses corresponding to a butterfly. For the same reason as above, one - of - two selectors are used to select either the normal address or the corresponding scrambled address.

4.4 Memory Clock and Gating Signals

The control unit has to generate many signals like Read memory, Write memory, load the data in data registers and so on. The pattern in which these various signals occur is shown as a flow chart in Fig. 4.1.

Since only a single word can be accessed from the memory at a time, the real and imaginary parts of the data for FFT have to be fetched one after another from the memory. Since two data are involved in each computation, a total of 4 accesses to memory are required for reading and 4 more for writing. Except for the last data in last iteration, writing the previous data is always followed by reading the next data and hence a burst of 8 pulses can be used as the memory clock with the first 4 pulses for writing and the next 4 pulses for reading.

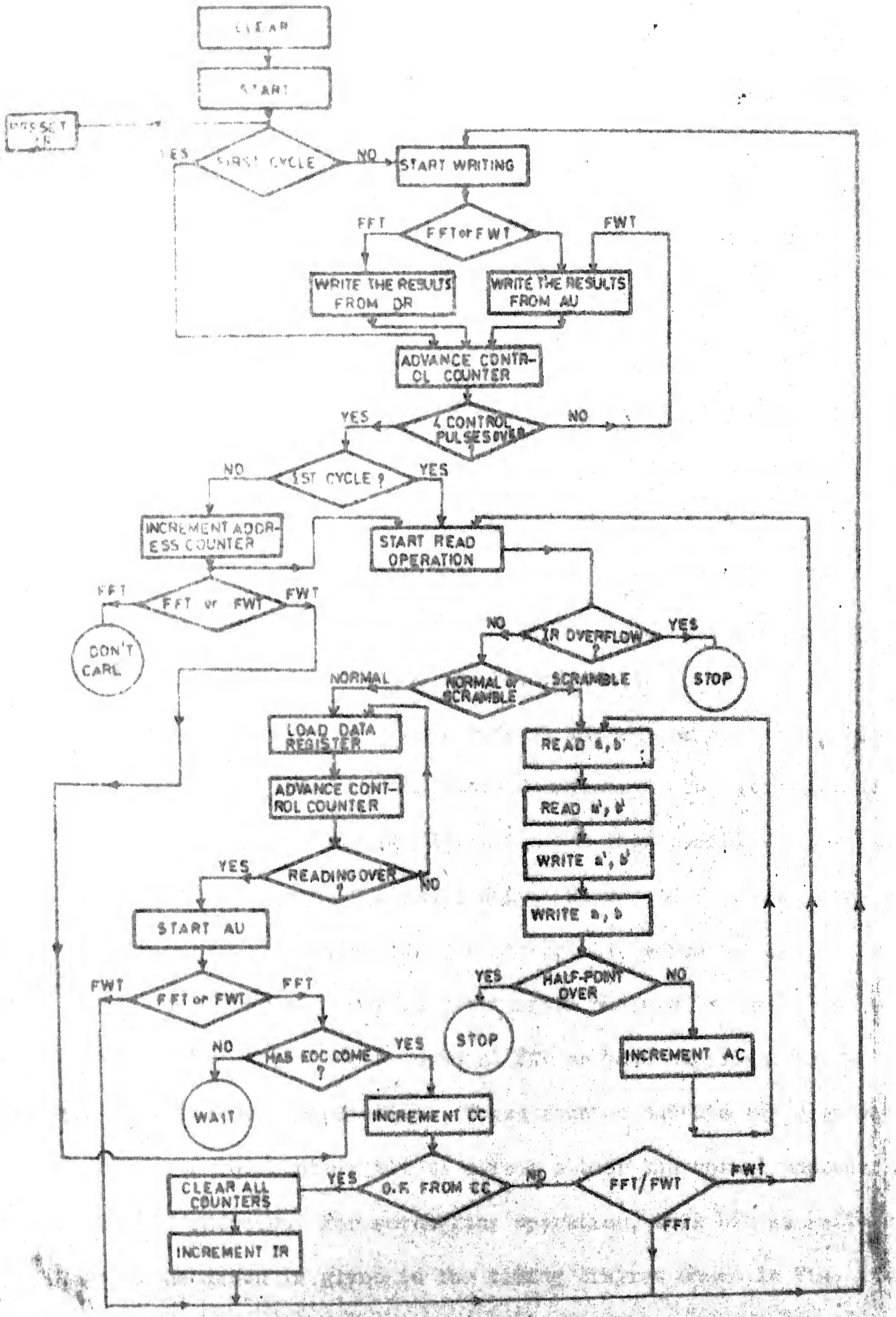
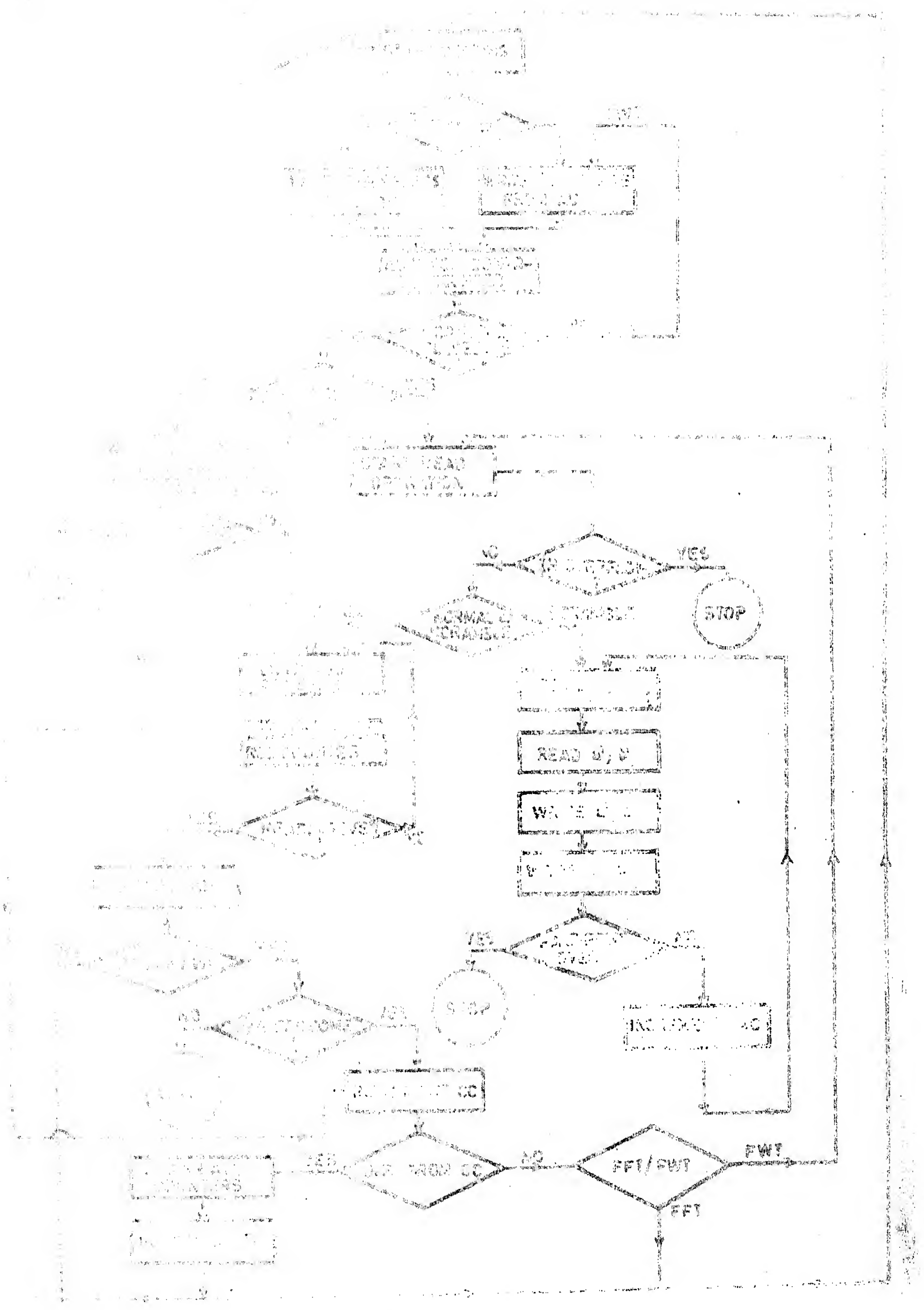


FIG.4.1 FLOW CHART FOR CONTROL UNIT



[illegible] FLOW CHART FOR CONTROL UNIT

There is a 'scramble' switch which starts the memory clock just as in normal operation. Since scrambling does not involve any AU operation, the memory clock is stopped only when the scrambling corresponding to half the total number of points is over.

While data is read into registers a,b,c, and d, the data should be directed to one of the registers at a time and this gating is done by the shift register 7495 with a set of NAND gates. For FWT, the data loaded in registers b and d are not used, and while writing, the resultant data need not be brought back to the registers as it can be directly written in the memory since they are available in parallel. So one - of - two selectors as shown in Fig. A2.15 are necessary to select either the FFT data or the FWT data before writing in memory. A single control bit is used to determine whether FFT or FWT is used, and this bit is realised by a SPDT switch.

Buffers are provided at various places where the fan-out exceeds the limit. In FFT, when the increment command is given, it takes some time for the increment counter and decoder to settle down and so some delay is necessary between the occurrence of increment command and the loading of the increment in increment registers. To provide this delay and to produce some very short pulses, the monostables are used in many places. The LED displays are provided to indicate that all the computations are over and all the scrambling operations are over.

There is a 'scramble' switch which starts the memory clock just as in normal operation. Since scrambling does not involve any AU operation, the memory clock is stopped only when the scrambling corresponding to half the total number of points is over.

While data is read into registers a,b,c, and d, the data should be directed to one of the registers at a time and this gating is done by the shift register 7495 with a set of NAND gates. For FWT, the data loaded in registers b and d are not used, and while writing, the resultant data need not be brought back to the registers as it can be directly written in the memory since they are available in parallel. So one - of - two selectors as shown in Fig. A2.15 are necessary to select either the FFT data or the FWT data before writing in memory. A single control bit is used to determine whether FFT or FWT is used, and this bit is realised by a SPDT switch.

Buffers are provided at various places where the fan-out exceeds the limit. In FFT, when the increment command is given, it takes some time for the increment counter and decoder to settle down and so some delay is necessary between the occurrence of increment command and the loading of the increment in increment registers. To provide this delay and to produce some very short pulses, the monostables are used in many places. The LED displays are provided to indicate that all the computations are over and all the scrambling operations are over.

4.5 Modularity

In the present context, modularity means that the basic design remains unchanged for any number of points provided it is a power of 2. Since the control unit for FFT and FWT is highly symmetrical and repetitive it can be made to act for various number of points. It can be easily made modular by a simple switch which causes the initial '1' to be loaded in different bits of the iteration register and also it selects the overflow of an appropriate bit of the computation counter. The 'Increment Logic' suggested also allows for the modular design as the increments and the pattern they follow are the same and all increments need not be used.

REFERENCES

1. J.W. Cooley and J.W. Tukey, 'An algorithm for the machine calculation of Complex Fourier Series', Math. of Comput., Vol. 19, pp. 297-301, April 1965.

2. John L. Shanks, 'Computation of Fast Walsh-Fourier Transform', IEEE Trans, on Computers, pp. 457-459, 1969.

3. I.J. Good, 'The interaction algorithm and practical Fourier analysis', Jour. Royal. Stat. Soc., Vol. B-20, pp. 361-372, 1958.

4. A.S. Kanade, 'Fast Fourier Transform Generator', a thesis submitted for the Dgree of Master of Technology to the Department of Electrical Engineering, I.I.T. Kanpur, July 1975.

5. G-AE Sub-Committee on Measurement concepts, 'What is Fast Fourier Transform', IEEE Trans. on Audio and Electro Acoustics, Vol. AU15, pp. 76-79, June 1967.

6. G.D. Bergland, 'FFT hardware implementation - an overview', IEEE Trans. on Audio and Electro Acoustics, Vol. AU-15, pp. 104-108, June 1967.

7. C. Yuen, 'Walsh Functions and Gray Code', Applications of Walsh Functions, 1971 Proceedings, Symposium.

APPENDIX

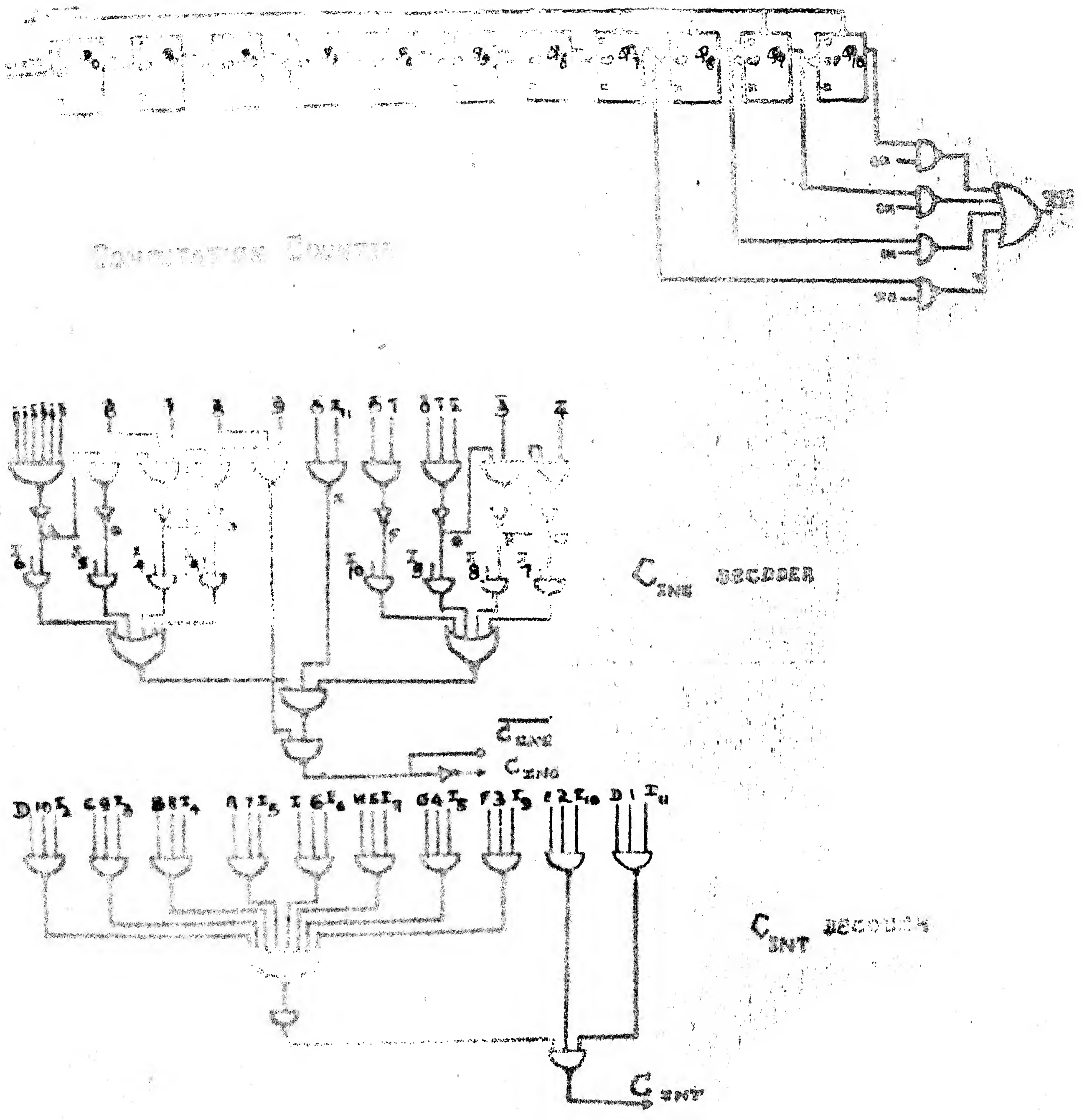
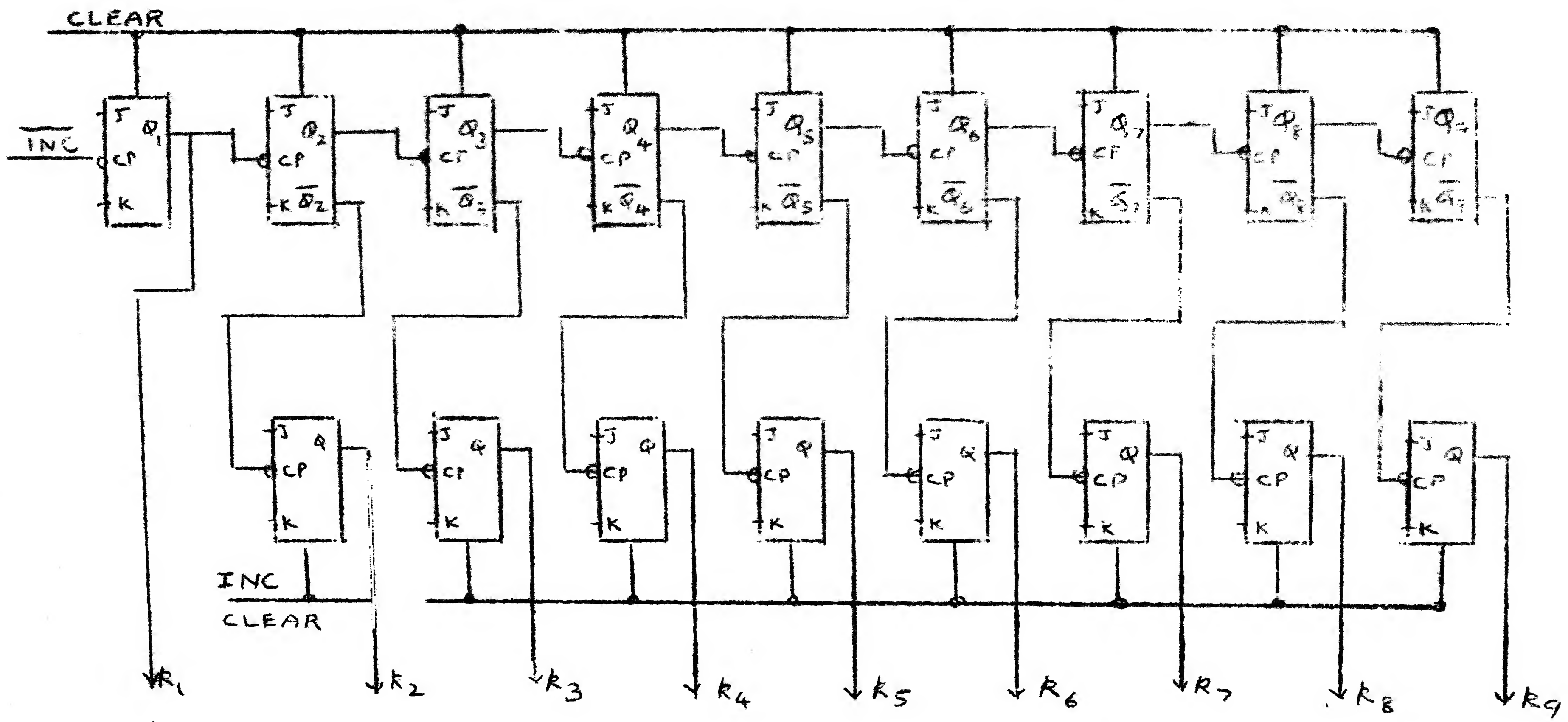
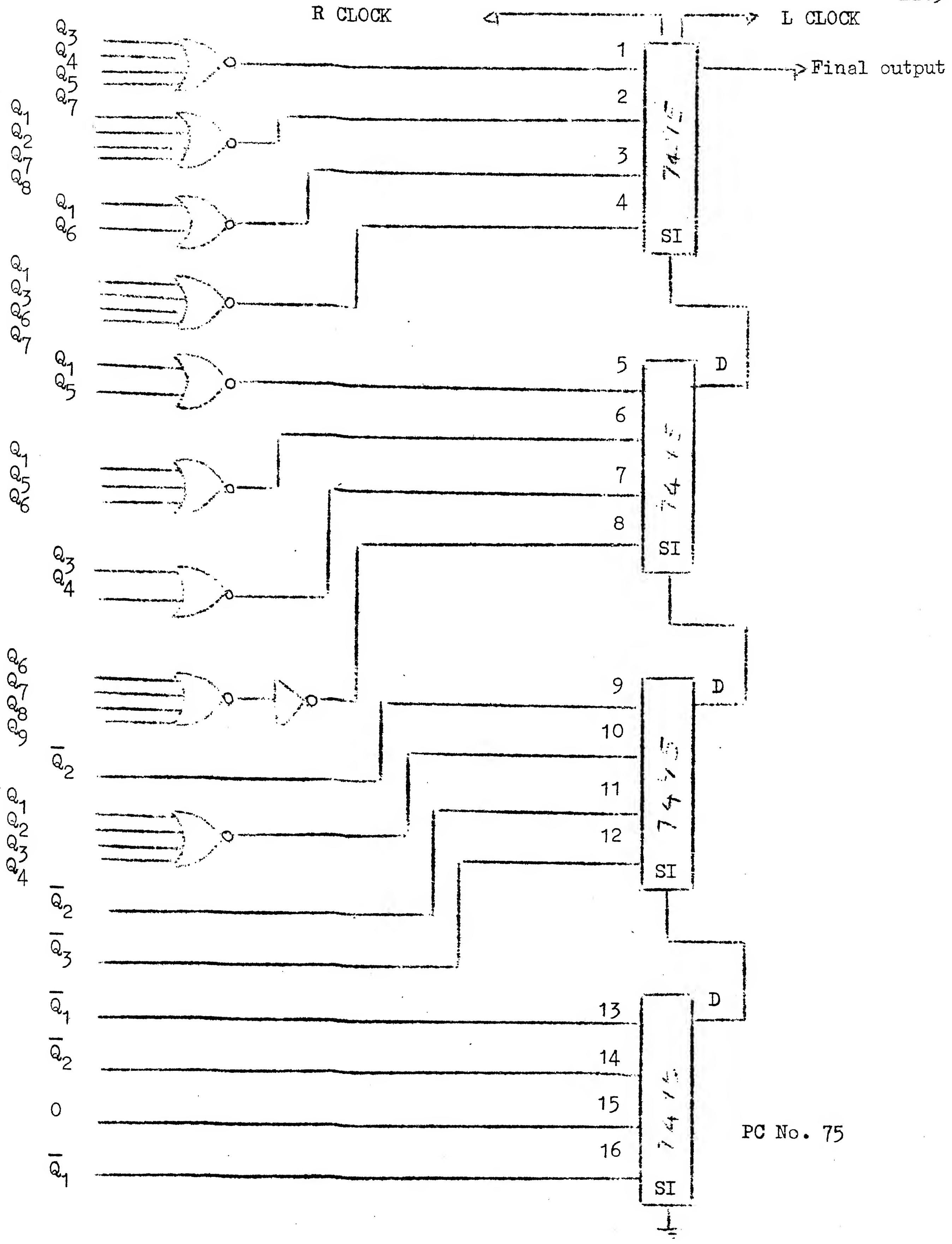


FIG A 8.1 COMPUTATION COUNTER AND C_{INT}, C_{INC} GENERATION



J = K = 1 in all flip-flops FC No 74

Fig. A2.2 INCREMENT COUNTER


R CLOCK
L CLOCK
Final output
Q_3 Q_4 Q_5 Q_7
Q_1 Q_2 Q_7 Q_8
Q_1 Q_6
Q_1 Q_3 Q_6 Q_7
Q_1 Q_5
Q_1 Q_5 Q_6
Q_3 Q_4
Q_6 Q_7 Q_8 Q_9
$\overline{Q}_2$
Q_1 Q_2 Q_3 Q_4
$\overline{Q}_2$
$\overline{Q}_3$
$\overline{Q}_1$
$\overline{Q}_2$
0
$\overline{Q}_1$
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
7495
7495
7495
7495
SI
SI
SI
SI
D
D
D

PC No. 75

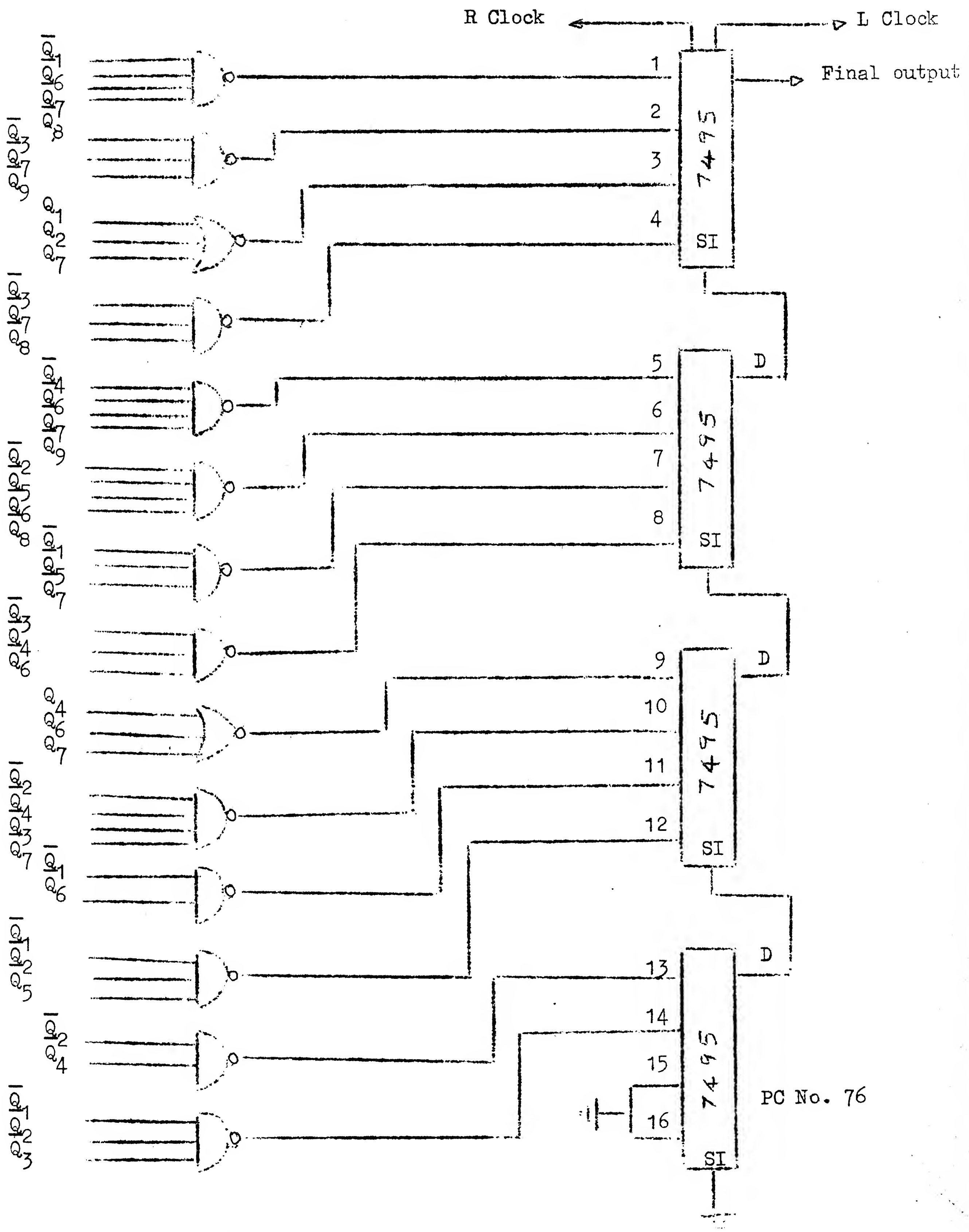
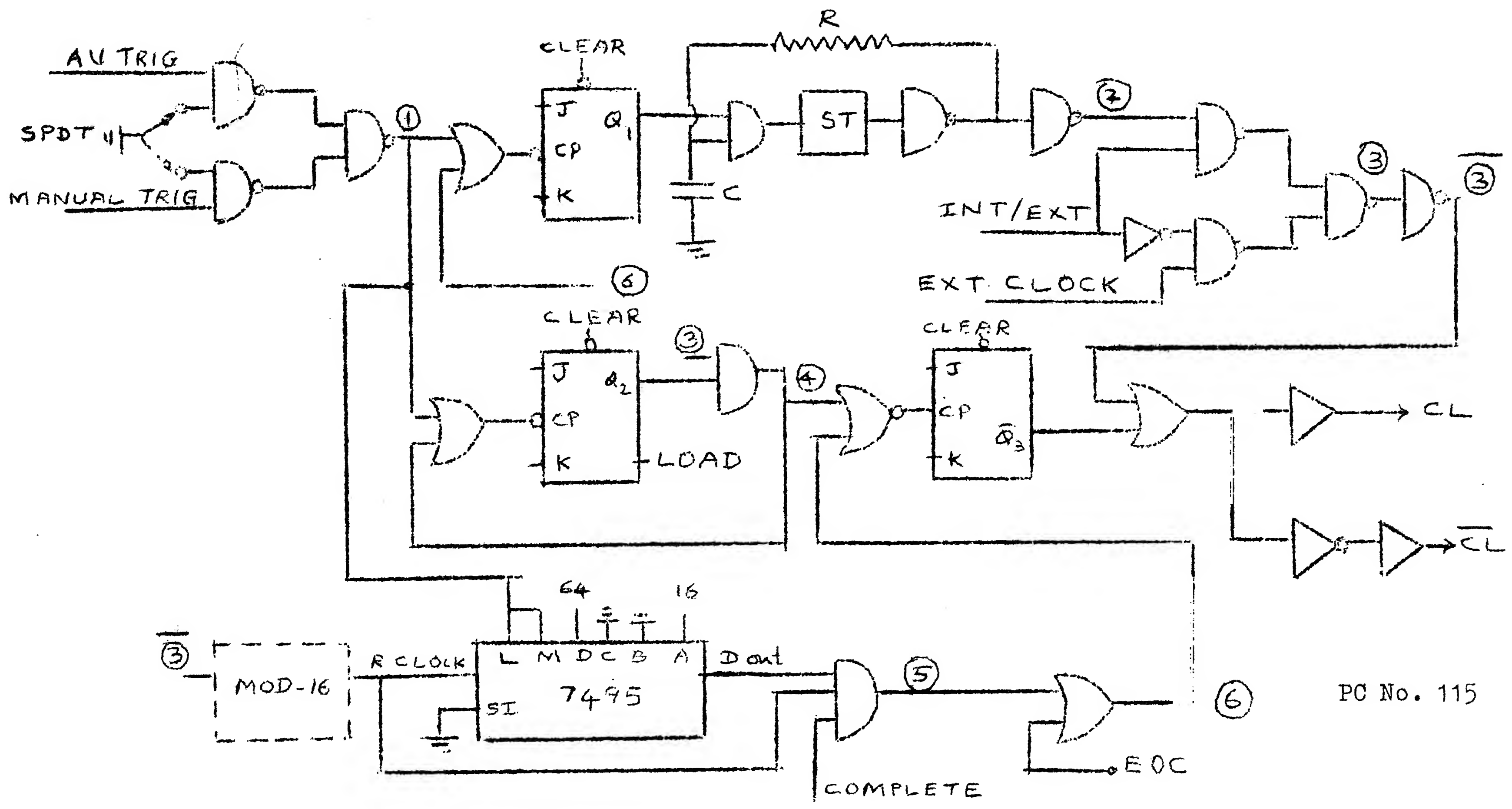
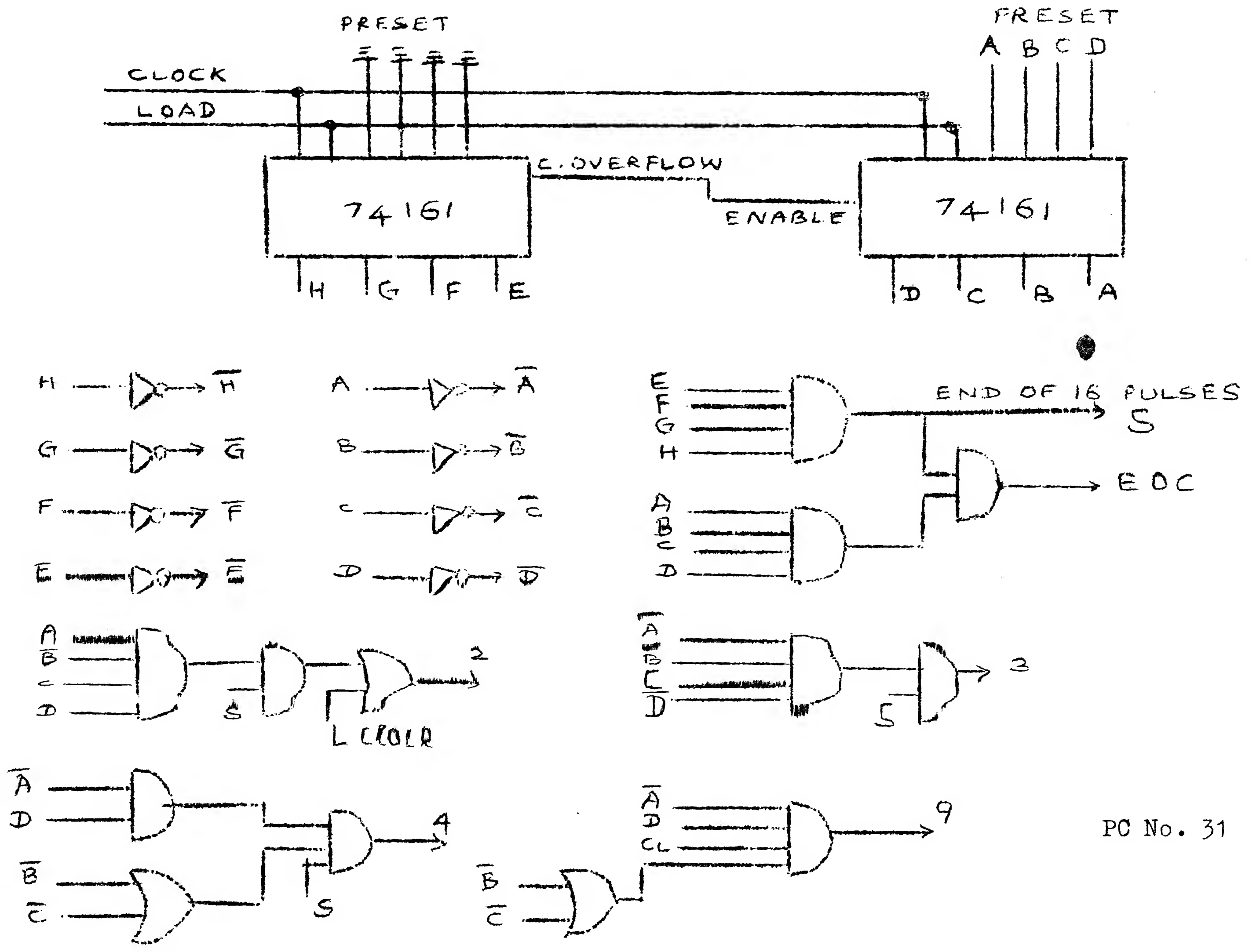


Fig. A2.4 WEIGHT GENERATION (IMAGINARY PART)



PC No. 115

Fig. A2.5a AU CLOCK GENERATOR



A = 0

B = No INC/

C = INC

D = INT

PC No. 31

Fig. A2.5b CLOCK COUNTER

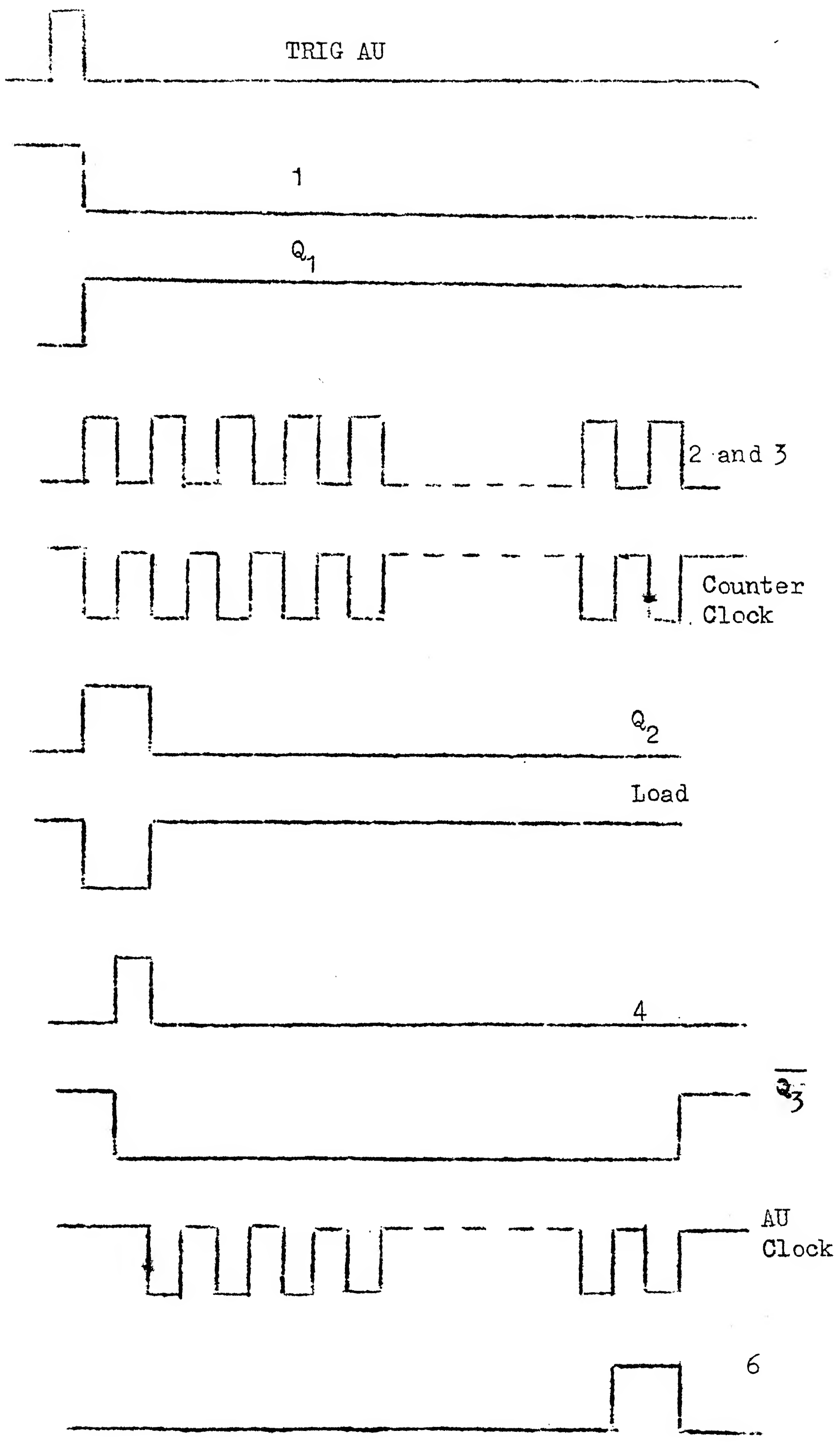


Fig. A2.6 TIMING DIAGRAM FOR CLOCK GENERATION

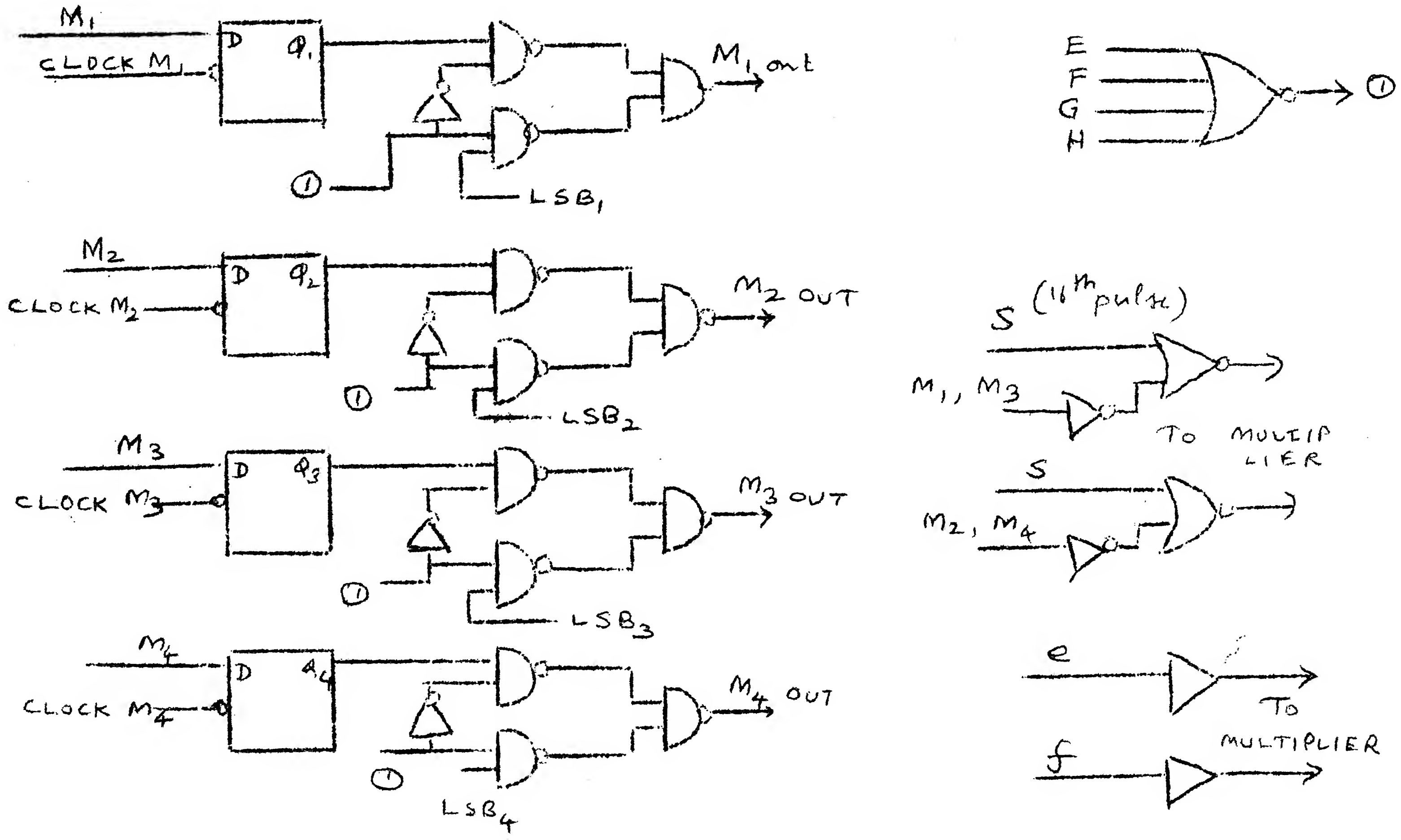


Fig. A2.7 BUFFER FOR MALTIPLIER

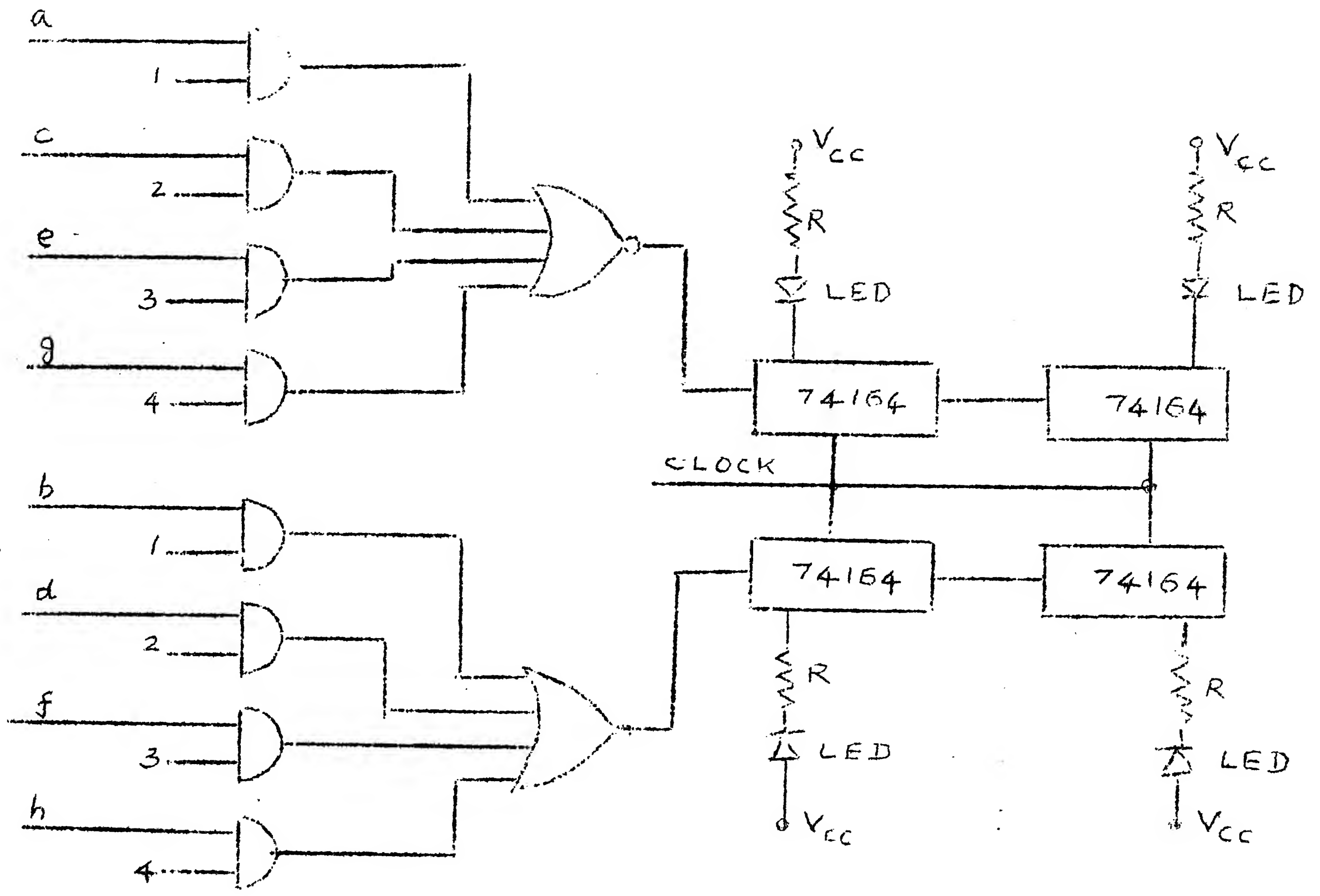


Fig. A2.8 DATA DISPLAY

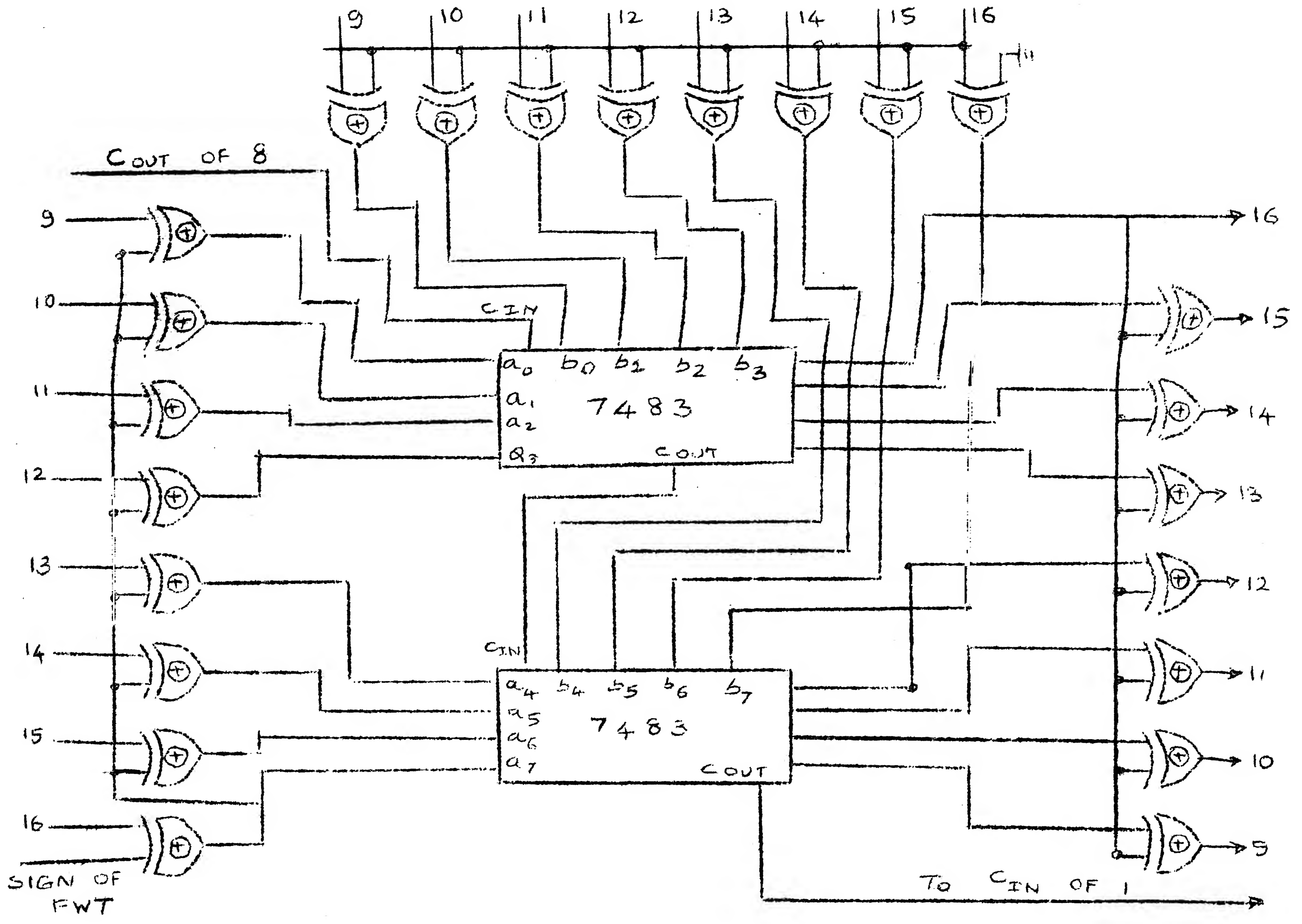
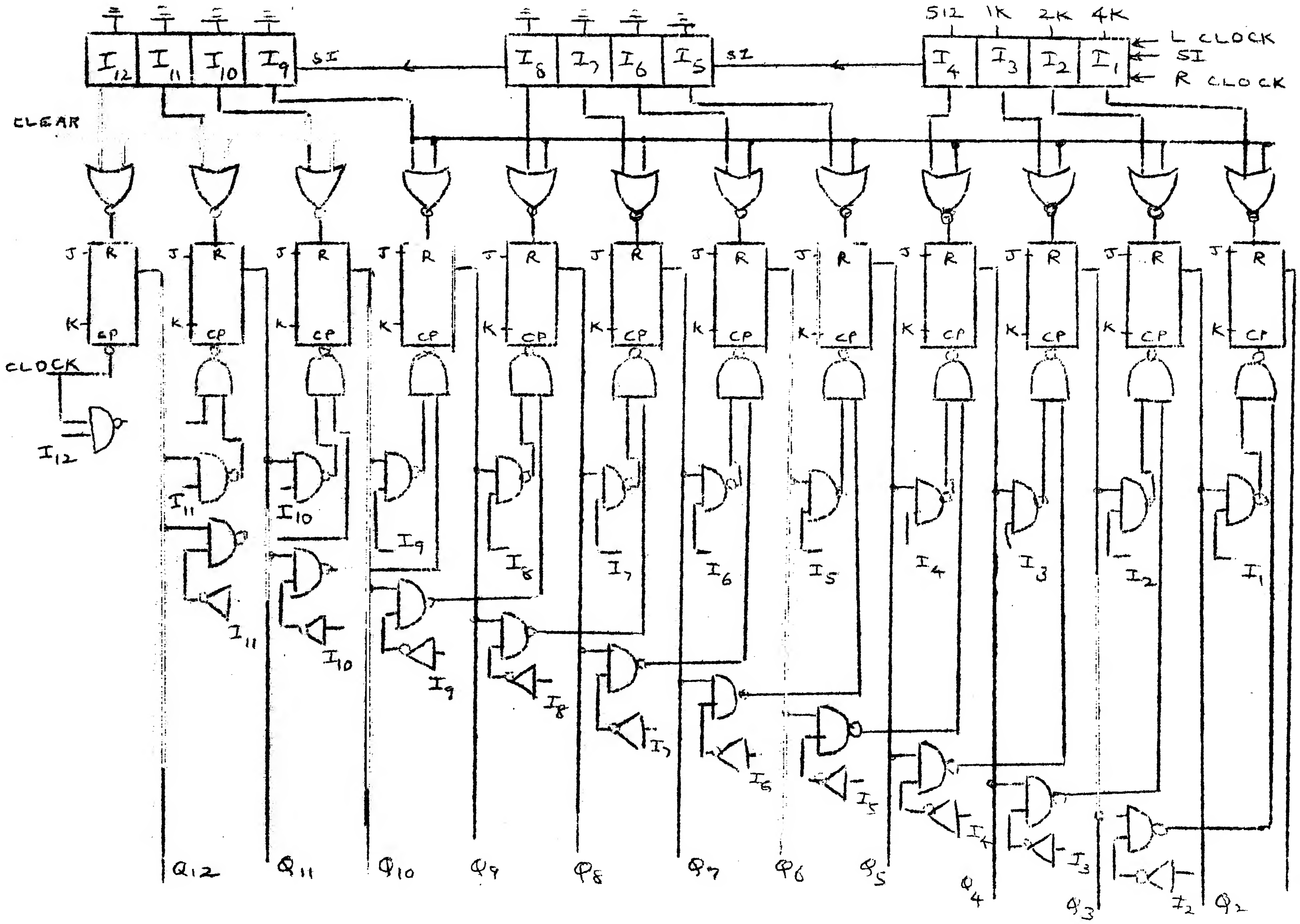


Fig. A2.9 FWT ARITHMETIC UNIT (2 CARDS)


512
1K
2K
4K
L CLOCK
SI
R CLOCK
I_1 I_2 I_3 I_4 I_5 I_6 I_7 I_8 I_9 I_{10} I_{11} I_{12}
SI
CLEAR
CLOCK
J
K
R
CP
Q
Q_2 Q_3 Q_4 Q_5 Q_6 Q_7 Q_8 Q_9 Q_{10} Q_{11} Q_{12}

A2.10 ITERATION REGISTER. ADDRESS COUNTER

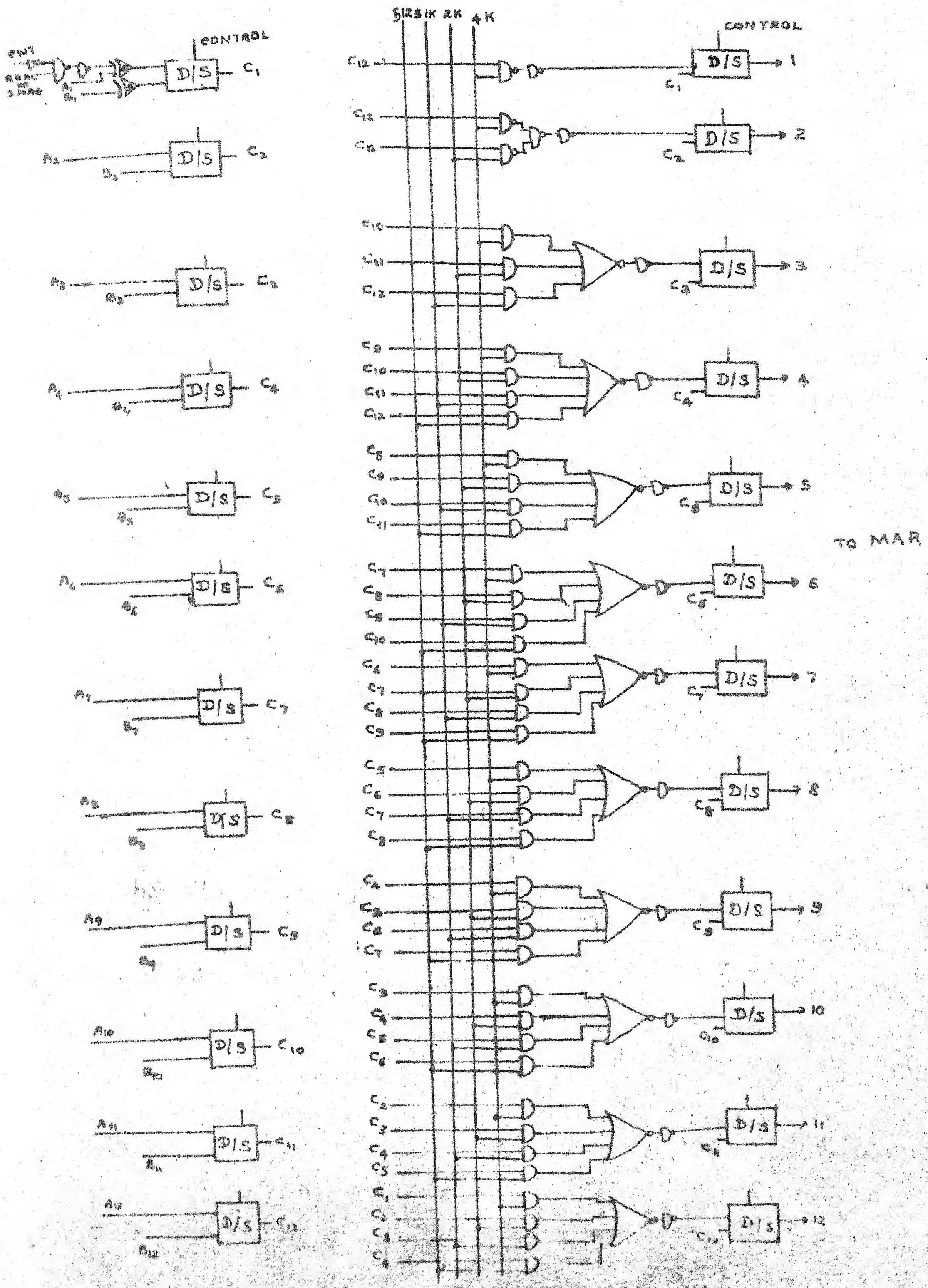


FIG. A20 ADDRESS SCRAMBLER

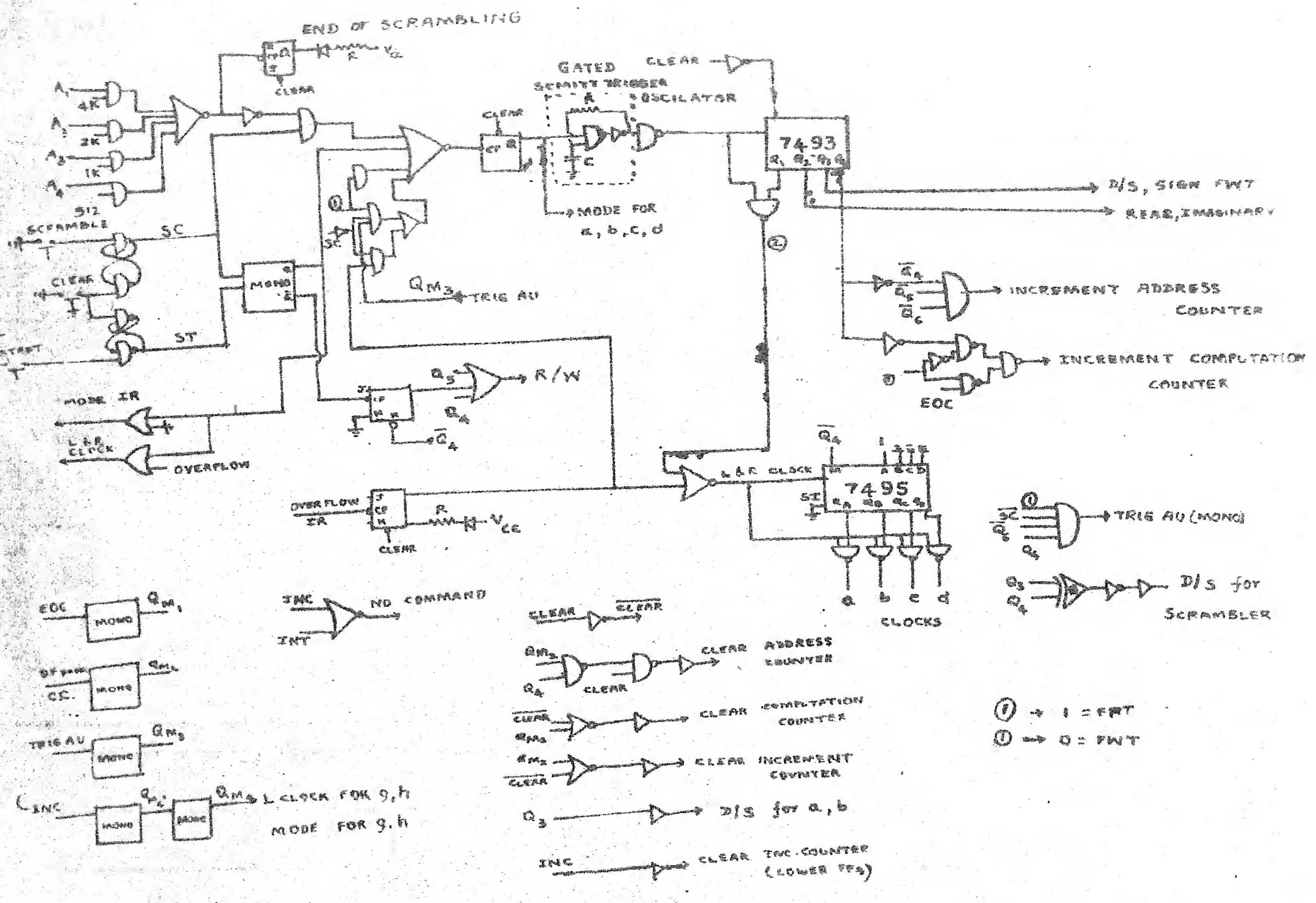


FIG A 2.13 MAIN CONTROL UNIT

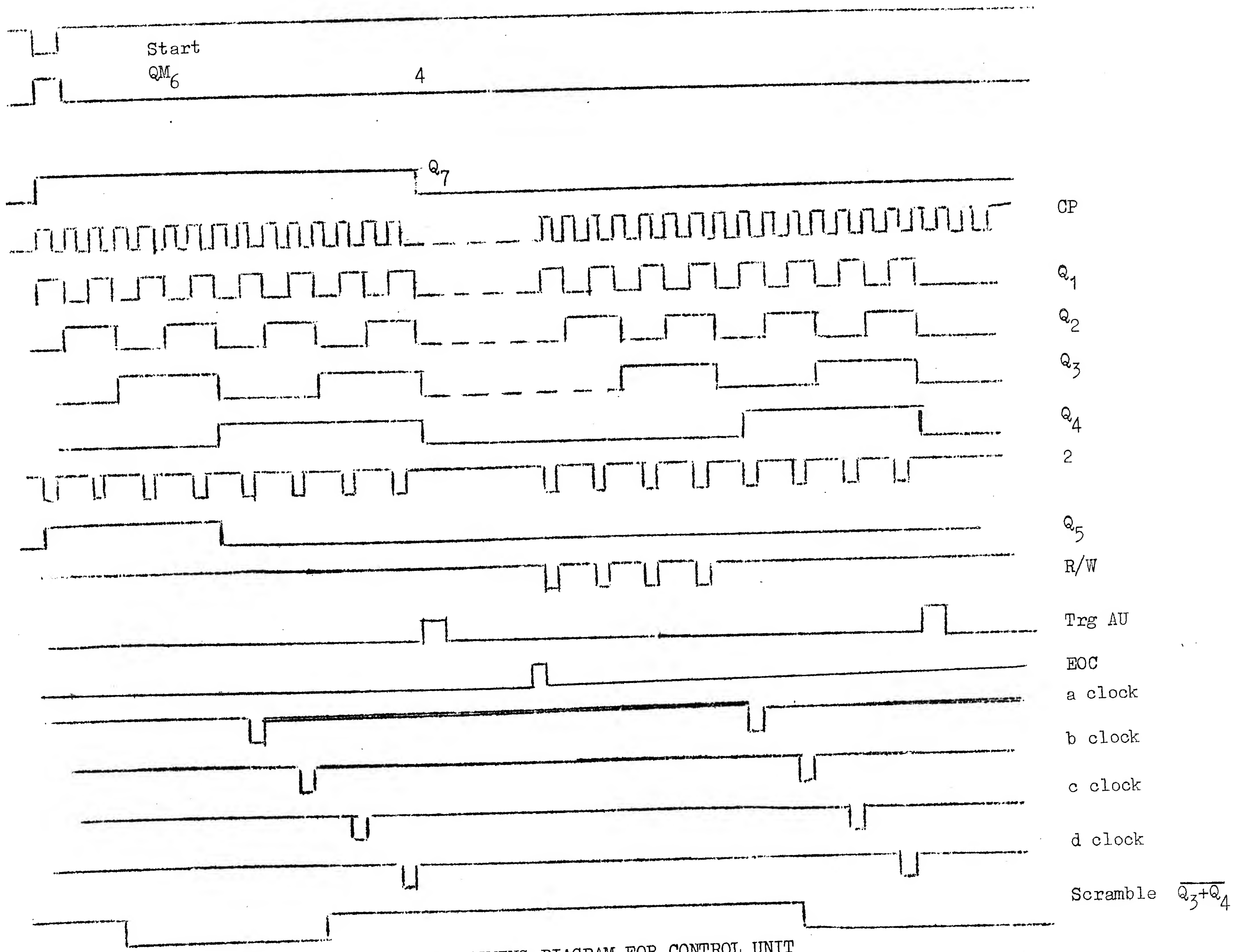


Fig. A2.14 TIMING DIAGRAM FOR CONTROL UNIT

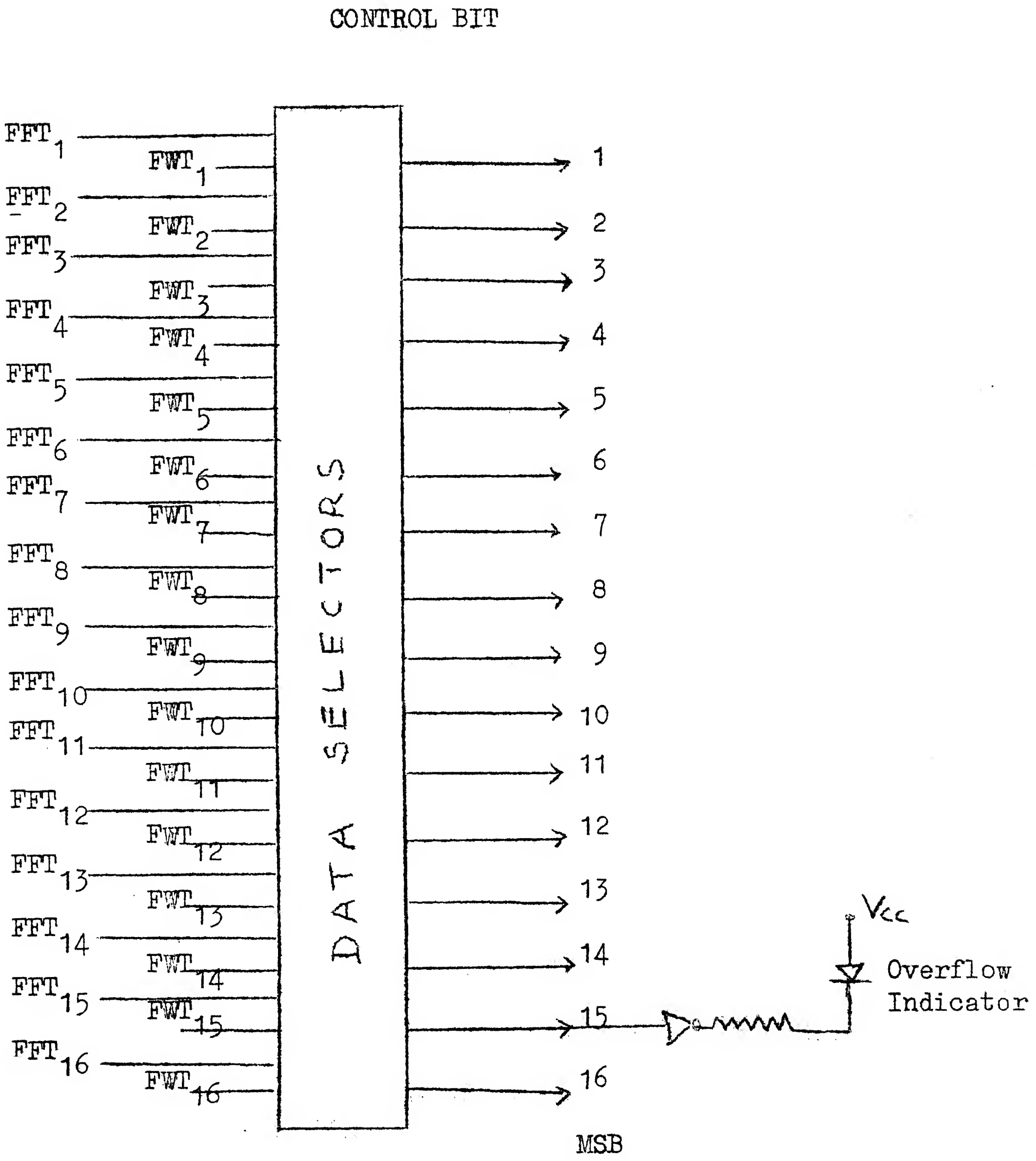


Fig. A2.15 FFT/FWT DATA SELECTOR